版权注意事项：
1、书籍版权归作者和出版社所有
2、本PDF仅限用于个人获取知识，进行私底下的知识交流
3、PDF获得者不得在互联网上以任何目的进行传播
4、如觉得书籍内容很赞，请购买正版实体书，支持作者
5、请于下载PDF后24小时内删除本PDF。

Broadview
www.broadview.com.cn



高等院校软件专业方向 系列教材

Web Services 应用开发

青岛东合信息技术有限公司　青岛海尔软件有限公司　编著



電子工業出版社
PUBLISHING HOUSE OF ELECTRONICS INDUSTRY
http://www.phei.com.cn

编　委　会

主　编： 邵峰晶

副主编： 韩敬海　　于忠清

特约策划人： 吕蕾

编　委： 曹宝香　　孙运彩　　崔文善

王成端　　薛庆文　　庞善臣

高仲合　　陈龙猛　　张　磊

吴海峰　　刘宝旨



高等院校软件专业方向 系列教材

Web Services 应用开发

青岛东合信息技术有限公司　青岛海尔软件有限公司　编著

電子工業出版社
Publishing House of Electronics Industry
北京·BEIJING

内容简介

本书分别介绍了XML、DOM、DTD、Schema、SOAP、XSL、Web Services等技术知识。其中，详细介绍了XML的基本结构和XML的格式控制；Java处理XML的方法和技巧；以及介绍了如何使用Java Web服务开发工具包JWSDP（Java Web Services Developer Pack）来开发Web服务；此外还介绍了如何使用开源工具包CXF开发Web Services应用。

本书重点突出、偏重应用，结合理论篇的实例和实践篇对案例的讲解、剖析及实现，使读者能迅速理解并掌握知识，全面提高动手能力。

本书可作为本科计算机科学与技术、软件外包专业、高职高专计算机软件、计算机网络、计算机信息管理、电子商务和经济管理等专业的程序设计课程的教材。

未经许可，不得以任何方式复制或抄袭本书之部分或全部内容。
版权所有，侵权必究。

图书在版编目（CIP）数据

Web Services应用开发 / 青岛东合信息技术有限公司，青岛海尔软件有限公司编著. —北京：电子工业出版社，2011.7
高等院校软件专业方向系列教材
ISBN 978-7-121-13545-3

Ⅰ. ①W… Ⅱ. ①青… ②青… Ⅲ. ①互联网络—网络服务器 Ⅳ. ①TP368.5

中国版本图书馆CIP数据核字（2011）第085637号

责任编辑：张月萍
文字编辑：王　静
印　　刷：三河市鑫金马印装有限公司
装　　订：
出版发行：电子工业出版社
北京市海淀区万寿路173信箱　　邮编：100036
开　　本：787×1092　1/16　　印张：19.75　　字数：570千字
印　　次：2011年7月第1次印刷
定　　价：39.00元

凡所购买电子工业出版社图书有缺损问题，请向购买书店调换。若书店售缺，请与本社发行部联系，联系及邮购电话：（010）88254888。
质量投诉请发邮件至zlts@phei.com.cn，盗版侵权举报请发邮件到dbqq@phei.com.cn。
服务热线：（010）88258888。

前　言

随着 IT 产业的迅猛发展，企业对应用型人才的需求越来越大。“全面贴近企业需求，无缝打造专业实用人才”是目前高校计算机专业教育的革新方向。

该系列教材是面向高等院校软件专业方向的标准化教材。教材研发充分结合软件企业的用人需求，经过了充分的调研和论证，并充分参照多所高校一线专家的意见，具有系统性、实用性等特点。旨在使读者在系统掌握软件开发知识的同时，着重培养其综合应用能力和解决问题的能力。

该系列教材具有如下几个特色。

1. 以应用型人才为导向来培养学生

强调实践：本系列教材以应用型软件及外包人才为培养目标，在原有体制教育的基础上对课程进行了改革，强化“应用型”技术的学习。使学生在经过系统、完整的学习后能够达到如下要求：

- 具备软件开发工作所需的理论知识和操作技能，能熟练进行编码工作，并掌握软件开发过程的规范；
- 具备一定的项目经验，包括代码的调试、文档编写、软件测试等内容；
- 相当于一年的软件开发经验。

2. 以实用技能为核心来组织教学

二八原则：遵循企业生产过程中的“二八原则”，即企业生产过程中 80%的时间在使用 20%的核心技术，强调核心教学，即学生在学校用 80%的学习时间来掌握企业中所用到的核心技术，从而保证对企业常用技术的掌握。教材内容精而专，同时配以知识拓展和拓展练习，以满足不同层次的教学和学习需求。

3. 以新颖的教材架构来引导学习

自成体系：本系列教材采用的教材架构打破了传统的以知识为标准编写教材的方法，采用“全真案例”和“任务驱动”的组织模式。

- **理论篇**：即最小教学集，包含了“二八原则”中提到的常用技术，以任务驱动引导知识点的学习，所选任务不但典型、实用，而且具有很强的趣味性和可操作性，引导学生循序渐进地理解和掌握这些知识和技能，培养学生的逻辑思维能力，掌握利用开发语言进行程序设计的必备知识和技巧。
- **实践篇**：多点于一线，以完整的具体案例贯穿始终，力求使学生在动手实践的过程中，加深课程内容的理解，培养学生独立思考和解决问题的能力，并配备相关知识的拓展讲解和拓展练习，拓宽学生的知识面。
- **结构灵活**：本系列教材在内容设置上借鉴了软件开发中“低耦合高内聚”的设计理念，组织架构上遵循软件开发中的 MVC 理念，即在课程的实施过程中各高校可根据自身的实际情况（课程配比、时间安排、学生水平、教学情况等），在保证最小教学集的前提下可对整个课程体系进行横向（章节内容）、纵向（章节）裁剪。

4. 提供全面的教辅产品来辅助教学实施

为充分体现“实境耦合”的教学模式，方便教学实施，另外还开发了可配套使用的项目实训教材和全套教辅产品，可供各院校选购：

项目篇：多线于一面，以辅助教材的形式，提供适应当前课程（及先行课程）的综合项目，遵循软件开发过程，进行讲解、分析、设计、指导，注重工作过程的系统性，培养学生解决实际问题的能力，是实施“实境”教学的关键环节。

立体配套：为适应教学模式和教学方法的改革，本系列教材提供完备的教辅产品，主要包括教学指导、实验指导、电子课件、习题集、实践案例等内容，并配以相应的网络教学资源。教学实施方面，提供全方位的解决方案（课程体系解决方案、实训解决方案、教师培训解决方案和就业指导解决方案等），以适应软件开发教学过程的特殊性。

本系列教材由青岛东合信息技术有限公司和青岛海尔软件有限公司共同研制，历时两年，参与编著的有韩敬海、丁春强、赵克玲、高峰、张幼鹏、张旭平、孙更新、宾晟、胡芳、邵峰晶、于忠清、张磊、曹宝香、崔文善、王成端等。本书的特约策划人为吕蕾女士。参与本书编写工作的还有：青岛农业大学、潍坊学院、青岛大学、山东科技大学、曲阜师范大学、济宁学院、中国海洋大学、济宁医学院等高校，期间得到了各合作院校专家及一线教师的大力支持和协作。在此技术丛书出版之际要特别感谢给予我们开发团队大力支持和帮助的领导及同事，感谢合作院校的师生给予我们的支持和鼓励，更要感谢开发团队每一位成员所付出的艰辛劳动。如有意见或建议，请发邮件至 dh_iTeacher@126.com。

iTeacher@ 教研组

2011 年 6 月

目　录

理论篇 1

第 1 章　XML 概述 2

1.1　XML 概述 4
1.1.1　XML 的历史 4
1.1.2　XML 的基本特征 5
1.1.3　XML 定义 6
1.2　XML 文档结构 8
1.2.1　序言 9
1.2.2　文档元素 11
1.3　XML 文档内容 11
1.3.1　XML 元素 11
1.3.2　XML 属性 14
1.3.3　注释 16
1.3.4　字符引用和实体引用 16
1.3.5　CDATA 节 18
1.3.6　处理指令 19
1.4　XML 文档规则 21
1.4.1　格式良好的 XML 文档规则 21
1.4.2　格式良好的 XML 文档 23
1.4.3　有效的 XML 文档 23
小结 24
练习 25

第 2 章　XML 文档类型定义 26

2.1　DTD 概述 28
2.1.1　DTD 简介 28
2.1.2　DTD 声明 29
2.1.3　内部 DTD 29
2.1.4　外部 DTD 30
2.2　DTD 语法 32

2.2.1 元素声明 32
2.2.2 属性声明 35
2.2.3 实体 41
小结 43
练习 44

第 3 章 XML 模式定义 46

3.1 命名空间 48
3.1.1 概述 48
3.1.2 创建命名空间 49
3.2 Schema 概述 51
3.2.1 Schema VS DTD 52
3.2.2 Schema 文档结构 53
3.3 Schema 中的数据类型 54
3.3.1 简单类型 54
3.3.2 复合类型 58
3.4 元素声明 64
3.4.1 简单元素 64
3.4.2 复合元素 65
3.4.3 匿名类型定义 67
3.5 属性声明 68
3.5.1 属性 68
3.5.2 属性组 71
3.6 模式重用 73
3.6.1 include 元素 73
3.6.2 import 元素 75
小结 78
练习 79

第 4 章 XML 显示——XSL 81

4.1 级联样式表 CSS 83
4.1.1 CSS 概述 83
4.1.2 CSS 语法 83
4.1.3 XML 与 CSS 关联 84
4.2 XSL 基础 86
4.2.1 XSL 86
4.2.2 XSLT 87
4.2.3 XPath 88

4.2.4 XPath 表达式 90
4.2.5 XPath 定位路径 95
4.3 XSL 文档 97
4.3.1 XSL 文档结构 97
4.3.2 XSL 元素 100
4.4 多样式表 115
4.4.1 <xsl:import>元素 115
4.4.2 <xsl:include>元素 117
小结 119
练习 119

第 5 章 运用 DOM 处理 XML 122

5.1 文档对象模型 124
5.1.1 概述 124
5.1.2 DOM 文档结构 124
5.2 解析 XML 127
5.2.1 XML 的解析方式 127
5.2.2 解析 XML 的 API 128
5.3 DOM 编程 131
5.3.1 Java DOM 的 API 131
5.3.2 DOM 应用 134
5.3.3 DOM 解析器属性 143
小结 147
练习 147

第 6 章 SAX 与 JDOM 150

6.1 使用 SAX 解析 XML 152
6.1.1 SAX 的实现机制 152
6.1.2 SAX 中的事件 152
6.1.3 Java SAX 的 API 155
6.1.4 SAX 编程 156
6.2 使用 JDOM 解析 XML 168
6.2.1 JDOM 概述 168
6.2.2 JDOM 的 API 169
6.2.3 JDOM 编程 171
6.3 SAX、DOM 和 JDOM 技术的比较 175
小结 176
练习 177

第 7 章 Web Services 概述 179

7.1 Web Services 简介 180
7.1.1 引言 180
7.1.2 Web Services 的特点 181
7.1.3 Web Services 的组成 182
7.1.4 Web Services 的优势与局限 184
7.2 Web Services 体系结构 186
7.2.1 Web Services 理论模型 186
7.2.2 Web Services 协议 188
7.2.3 Web Services 通信模型 190
7.2.4 实现 Web Services 192
小结 195
练习 195

第 8 章 SOAP 197

8.1 SOAP 概述 199
8.1.1 SOAP 介绍 199
8.1.2 SOAP 消息结构 200
8.1.3 SOAP 消息交换模型 205
8.1.4 SOAP 应用模式 208
8.1.5 SOAP 与 CORBA、COM/DCOM 的区别 210
8.2 JAXM 元素 211
8.2.1 SOAPElement 212
8.2.2 SOAPMessage 215
8.2.3 SOAPPart 216
8.2.4 SOAPEnvelope 216
8.2.5 SOAPHeader 和 SOAPHeaderElement 217
8.2.6 SOAPBody 和 SOAPBodyElement 218
8.2.7 SOAPFault 218
8.3 编写 SOAP 客户端 219
8.3.1 SOAP 连接 219
8.3.2 构建消息 221
8.3.3 编写 SOAP 客户端 222
8.4 编写 SOAP 服务器 225
8.4.1 JAXMServlet 225
8.4.2 编写 SOAP 服务器 227
8.4.3 部署测试 229

小结 229
练习 230

第 9 章 WSDL 和 UDDI 231

9.1 WSDL 233
9.1.1 WSDL 概述 233
9.1.2 WSDL 文档结构 234
9.1.3 WSDL 绑定 239
9.2 UDDI 241
9.2.1 UDDI 注册中心 241
9.2.2 UDDI 数据结构 242
9.2.3 UDDI API 243
9.2.4 WSDL 映射到 UDDI 245
小结 251
练习 252

第 10 章 RPC 253

10.1 RPC 概述 255
10.1.1 JAX-RPC 简介 255
10.1.2 JAX-RPC 应用程序体系结构 256
10.2 映射 Java 数据类型 257
10.2.1 一般数据类型 257
10.2.2 数组 259
10.2.3 映射为 JavaBean 260
10.3 创建基于 JAX-RPC 的 Web Service 261
10.3.1 下载并安装 JWSDP 2.0 261
10.3.2 wscompile 和 wsdeploy 工具 264
10.3.3 创建 JAX-RPC 服务器端 266
10.3.4 创建 JAX-RPC 客户端 270
小结 276
练习 276

理论篇 277

实践 1 SOAP 278

实践指导 278
实践 1.G.1 278
知识拓展 284

SOAP 安全性 284
拓展练习 287
练习 1.E.1 287

实践 2　CXF 应用 288

实践指导 288
实践 2.G.1 ..... *288*
知识拓展 296
Web Service 框架 296
拓展练习 298
练习 2.E.1 298

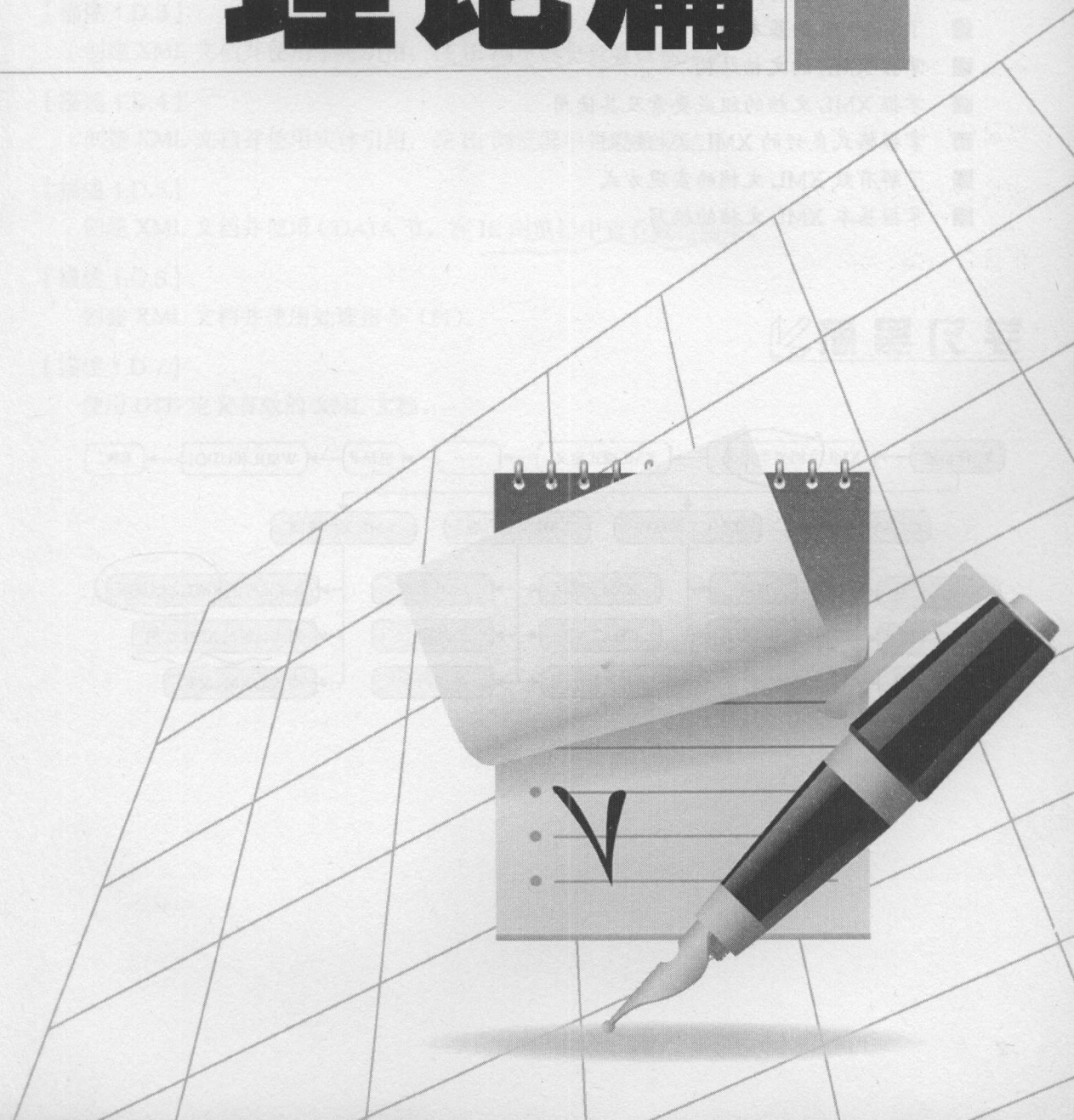
附录 A　XSD 元素 299

附录 B　XSD 限定 300

附录 C　XPath 函数库 301

附录 D　XSL 元素 303

理论篇

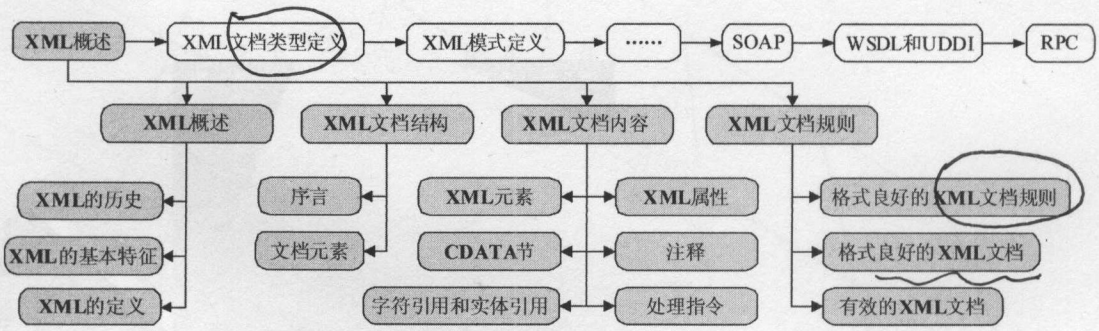


第 1 章　XML 概述

本章目标

- 了解 XML 的历史
- 了解 XML 的特点
- 了解 XML 的基本特征
- 掌握 XML 的文档结构
- 掌握 XML 文档的组成要素及其使用
- 掌握格式良好的 XML 文档规则
- 了解有效 XML 文档的实现方式
- 掌握基本 XML 文档的编写

学习导航



任务描述

【描述 1.D.1】

使用 XML 描述学生信息，并使用 IE 浏览器查看数据结果。

【描述 1.D.2】

创建含有中文字符的 XML 文件，并使用 IE 浏览器查看数据结果。

【描述 1.D.3】

创建 XML 文档并使用字符引用，在 IE 浏览器中查看数据结果。

【描述 1.D.4】

创建 XML 文档并使用实体引用，在 IE 浏览器中查看数据结果。

【描述 1.D.5】

创建 XML 文档并使用 CDATA 节，在 IE 浏览器中查看数据结果。

【描述 1.D.6】

创建 XML 文档并使用处理指令（PI）。

【描述 1.D.7】

使用 DTD 定义有效的 XML 文档。

1.1 XML 概述

XML（eXtensible Markup Language，可扩展标记语言）是一种元标记语言，是 Internet 环境中跨平台的、依赖于内容的技术，也是当今处理分布式结构信息的有效工具。

1.1.1 XML 的历史

XML 有两个先驱：SGML 和 HTML，这两个语言都是非常成功的标记语言，但是都具有一些与生俱来的缺陷。XML 正是为了解决它们的不足而诞生的。

1. SGML

早在 Web 未发明之前，SGML（Standard Generalized Markup Language，标准通用标记语言）就已存在，正如它的名称所言，SGML 是国际上定义电子文件结构和内容描述的标准。SGML 具有非常复杂的文档结构，主要用于大量高度结构化数据的访问和其他各种工业领域，在分类和索引数据中非常有用。

虽然 SGML 的功能很强大，但是它不适用于 Web 数据描述，而且 SGML 软件的价格非常昂贵；另外，SGML 十分庞大，既不容易学，也不容易使用，在计算机上实现也十分困难；不仅如此，几个主要的浏览器厂商都明确拒绝支持 SGML，这无疑是 SGML 在网上传播遇到的最大障碍。鉴于这些因素，Web 的发明者——欧洲核子物理研究中心的研究人员根据当时（1989 年）的计算机技术，开发了 HTML。

2. HTML

1989 年 HTML 诞生，它抛弃了 SGML 的复杂庞大的缺点，继承了 SGML 很多优点。HTML 最大的特点是简单和跨平台。

HTML 是一种界面技术，它只使用了 SGML 中很少的一部分标记，例如 HTML 4.0 中只定义了 70 余种标记。为了便于在计算机上实现，HTML 规定的标记是固定的，即 HTML 语法是不可扩展的。HTML 这种固定的语法使它易学易用，在计算机上开发 HTML 的浏览器也十分容易。正是由于 HTML 的简单性，使得基于 HTML 的 Web 应用得到极大的发展。

3. XML 的产生

随着 Web 应用的不断发展，HTML 的局限性也越来越明显地体现了出来，如 HTML 无法描述数据、可读性差、搜索时间长等。人们又把目光转向 SGML，再次改造 SGML 使之适合现在的网络需求。随着先辈的努力，在 1998 年 2 月 10 日，W3C（World Wide Web Consortium，万维网联盟）公布 XML 1.0 标准，XML 诞生了。

XML 最初的设计目的是为了 EDI（Electronic Data Interchange，电子数据交换），确切地说是为电子数据交换提供一个统一的标准数据格式。

在 EDI 应用过程中，XML 展现了如下优势。

- 低成本，XML 不需要 VAN（Value-Added Network，增值网络）的高额费用，中小企

业也负担得起。

- 定制商业规则，XML 允许用户创建自己的商业规则和格式。
- 容易解释，EDI 需要使用翻译软件来得到贸易伙伴的信息格式，而 XML 通过免费下载的解析器就可以很容易地解释。
- 平台独立，XML 是跨平台的语言，不管是什么平台，都能进行数据交换。

当前的一些内容建设者们已经开始开发各种各样的 XML 扩展，比如数学标记语言 MathML、化学标记语言 CML 等。此外，一些著名的 IT 公司，如 Oracle、IBM 以及微软等都积极地投入人力与财力研发 XML 相关软件与服务支持，这无疑确定了 XML 在 IT 产业的重要地位。

1.1.2　XML 的基本特征

XML 是一个精简的 SGML 子集，它将 SGML 的丰富功能与 HTML 的易用性结合到 Web 的应用中。XML 保留了 SGML 的可扩展功能，这使 XML 从根本上有别于 HTML。XML 要比 HTML 强大得多，它不再是固定的标记，而是允许定义不限数量的标记来描述文档中的资料，允许嵌套的信息结构。HTML 只是 Web 显示数据的通用方法，而 XML 则提供了一个直接处理 Web 数据的通用方法。

XML 具有以下一些特点。

- XML 可以从 HTML 中分离数据，即能够在 HTML 文件之外将数据存储在 XML 文档中，这样可以使开发者集中精力使用 HTML 做好数据的显示和布局，并确保数据改动时不会导致 HTML 文件也需要改动，从而方便维护页面。XML 也能够将数据以“数据岛”的形式存储在 HTML 页面中，开发者依然可以把精力集中到使用 HTML 格式化和显示数据上去。
- XML 可用于交换数据。基于 XML 可以在不兼容的系统之间交换数据，计算机系统和数据库系统所存储的数据有 N^N 种形式，对于开发者来说，最耗时间的工作就是在遍布网络的系统之间交换数据。把数据转换为 XML 格式存储将大大减少交换数据时的复杂性，还可以使这些数据能被不同的程序读取。
- XML 可应用于 B2B 中。例如在网络中交换金融信息，目前 XML 正成为在遍布网络的商业系统之间交换信息所使用的主要语言，许多与 B2B 有关的完全基于 XML 的应用程序正在开发中。
- 利用 XML 可以共享数据。XML 数据以纯文本格式存储，这使得 XML 更易读、更便于记录、更便于调试，使不同系统、不同程序之间的数据共享变得更加简单。
- XML 可以充分利用数据。XML 是与软件、硬件和应用程序无关的，数据可以被更多的用户、设备所利用，而不仅仅限于基于 HTML 标准的浏览器。其他客户端和应用程序可以把 XML 文档作为数据源来处理，就像操作数据库一样，XML 的数据可以被各种各样的“阅读器”处理。

- XML 可以用于创建新的语言。比如，WAP 和 WML 语言都是由 XML 发展来的。WML（Wireless Markup Language，无线标记语言）是用于标识运行于手持设备上（比如手机）的 Internet 程序的工具，它就采用了 XML 的标准。

总之，XML 使用一个简单而又灵活的标准格式，为基于 Web 的应用提供了一个描述数据和交换数据的有效手段。但是，XML 并非是用来取代 HTML 的。HTML 着重于如何描述将文件显示在浏览器中，而 XML 与 SGML 相近，着重于如何将数据以结构化方式表示。就网页显示功能来说，HTML 比 XML 强，但就文件的应用范畴来说，XML 比 HTML 超出很多。XML 和 HTML 在各个方面上的对比如表 1-1 所示。

表 1-1　XML 和 HTML 对比

对比项	XML	HTML
可扩展性	可扩展，能够定义新的标记元素	不可扩展，标记元素都是固定的
侧重点	侧重于结构化地描述数据	侧重于如何显示数据
语法	语法严格，要求标记嵌套、配对和遵循 DTD 树形结构	不要求标记的嵌套、配对等，不要求标记之间具有一定的顺序
可读性	结构清晰、易于阅读	难于阅读
可维护性	易于维护	难于维护
数据和显示关系	数据描述与显示方式相分离，具有保值性	数据和显示整合为一体，不具保值性

1.1.3　XML 定义

XML 是元标记语言，定义了用于定义与其他特定领域有关的、语义的、结构化的标记语言句法的语言。可以从以下几个方面来定义 XML。

- XML 是一种类似于 HTML 的标记语言。
- XML 是用来描述数据的。
- XML 的标记不是在 XML 中预定义的，用户可以自定义标记。
- XML 使用文档类型定义（DTD）或者模式（Schema）来描述数据。

下述代码用于实现任务【描述 1.D.1】，使用 XML 描述学生信息，并使用 IE 浏览器查看数据结果。

【描述 1.D.1】 student.xml

```
<?xml version="1.0" ?>
<!-- File Name:student.xml -->
<students>
    <student sex = "male">
        <name>Tom</name>
        <age>14</age>
        <tel>88889999</tel>
```

```
    </student>
    <student sex = "female">
        <name>Rose</name>
        <age>16</age>
        <tel>66667777</tel>
    </student>
    <student sex = "male">
        <name>Jack</name>
        <age>15</age>
    </student>
</students>
```

上述代码只能让读者感性地认识 XML，并不能实现什么具体应用，它只是用 XML 标记存储信息的文件。

对于 XML 文件，可以用 IE 5.0 或以上版本的浏览器来查看其内容，如果在 IE 浏览器中打开一个 XML 文档，它将用颜色显示文档编码的根和子元素，如图 1-1 所示。

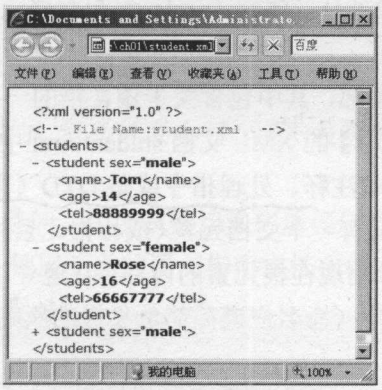


图 1-1　用 IE 浏览器查看 XML 文件结果

可以单击元素左边的加号（+）或减号（-）来展开或收缩元素的结构，如果想查看原始的 XML 源文件，就必须从浏览器菜单中选择“查看源文件”（View Source）命令，如果打开的 XML 文件是错误的，IE 浏览器就会报告错误，如图 1-2 所示。

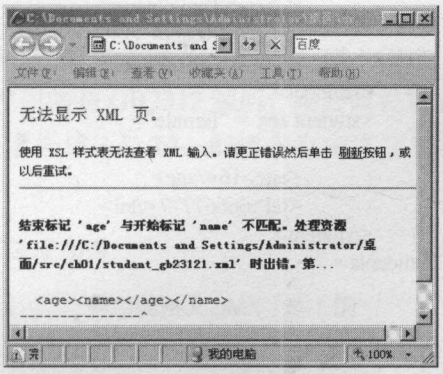


图 1-2　出错时的显示结果

XML 是自由的、可以扩展的，XML 标记（tag）并不是事先定义好的。在 XML 中，用户可以根据需要定义自己的标记以及文档结构。比如，上面例子中<students>、<name>、<age>等标记都是这个 XML 文档的作者创建出来的。而在 HTML 文档中必须使用 HTML 规范中定义好的标记，比如<P>、<hr>、<a>、等。

XML 可以从 HTML 中分离数据，即能够在 HTML 文件之外将数据存储在 XML 文档中，这样可以使开发者集中精力使用 HTML 做好数据的显示和布局，并确保数据改动时不会导致 HTML 文件也需要改动，从而方便维护页面。XML 也能够将数据以“数据岛”的形式存储在 HTML 页面中，开发者依然可以把精力集中到使用 HTML 格式化和显示数据上去。

1.2 XML 文档结构

XML 的语法规则既简单又严格，非常容易学习和使用。XML 文档使用了自描述的简单语法，如果熟悉 HTML，就会发现它的文档和 HTML 非常相似。

XML 文档有两个主要组成部分：序言（prolog）和文档元素（document element，即根元素）。

序言出现在 XML 文档的顶部，其中包含关于该文档的一些信息，有点儿像 HTML 文档中的<head>部分。在【描述 1.D.1】的 XML 文档 student.xml 中，序言包含了一个 XML 声明。序言也可以包含其他的元素，如注释、处理指令或是 DTD（文档类型定义）。

任何 XML 文档必须有且只有一个文档元素（或称根元素），用来包含可能有的其他内容。XML 文档中的所有内容都应该出现在根元素的内部。在遵守 XML 命名规则的前提下，用户可以为元素和属性选择任何名字（命名应遵循简单易懂的原则）。图 1-3 显示了 XML 文档的结构。

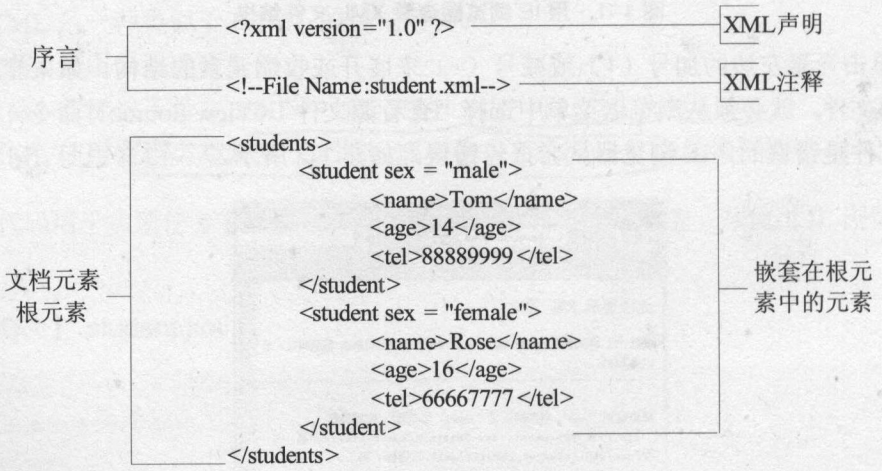


图 1-3 XML 文档的结构

1.2.1　序言

XML 序言由两行组成：

```
<?xml version="1.0" ?>
<!-File Name:student.xml-->
```

其中：

- 第一行是 XML 声明，表明这是一个 XML 文档，并且遵循的是 XML 1.0 版的规范。
- 第二行是注释。引入注释可以增强文档的可读性，XML 文档的注释是可选的。

XML 声明由“<?”开始，以“?>”结束。“<?”就表示该行是一个命令；在“<?”后面紧跟“xml”，表示该文件是一份 XML 文件（注意必须小写），这是处理指令的名称。在其后可声明 XML 的版本、采用的字符集等属性；在“<”和“?”之间、“?”和“>”之间，以及第一个“?”和“xml”之间不能有空格。在第二个“?”之前可以没有、也可以有一个或多个空格。

XML 声明语句中通常可以有如下属性。

- version：该属性是必需的，且必须要小写，用于表明 XML 的版本，解析器对不同的版本的解析会有区别。
- encoding：该属性是可选的，用于表明该文档所使用的字符编码方式。XML 支持多种字符集类型，例如，使用下面的语句指明文档中的字符编码方式为 GB2312 编码。

```
<?xml version="1.0" encoding="GB2312" ?>
```

- standalone：该属性定义了是否可以在不读取任何其他文件的情况下处理该文档，其属性值可以是 yes 或 no，如果 XML 文档没有引用任何其他文件，则可以指定 standalone="yes"，否则 standalone="no"。standalone 的默认值是 no。

注意　如果同时设置了 encoding 和 standalone 属性，standalone 属性要位于 encoding 属性之后。

对于含有中文字符的 XML，可以采用“Unicode”或“GB2312”等支持中文字符的编码来表示，如果文档中的字符使用的是 GB2312 编码，则必须设置 encoding 属性值为 GB2312。

下述代码用于实现任务【描述 1.D.2】，创建含有中文字符的 XML 文件，并查看数据结果。

【描述 1.D.2】 student_gb2312.xml

```
<?xml version="1.0" encoding="GB2312" ?>
<学生花名册>
    <学生 性别 = "男">
```

```
        <姓名>汤米</姓名>
        <年龄>14</年龄>
        <电话>88889999</电话>
    </学生>
    <学生 性别 = "女">
        <姓名>罗斯</姓名>
        <年龄>16</年龄>
        <电话>66667777</电话>
    </学生>
    <学生 性别 = "男">
        <姓名>杰克</姓名>
        <年龄>15</年龄>
    </学生>
</学生花名册>
```

上述 XML 代码中，由于存在中文字符，需要在 XML 声明中引入 encoding 属性，并设置属性值为“GB2312”，否则将会显示字符编码错误，如图 1-4 所示。

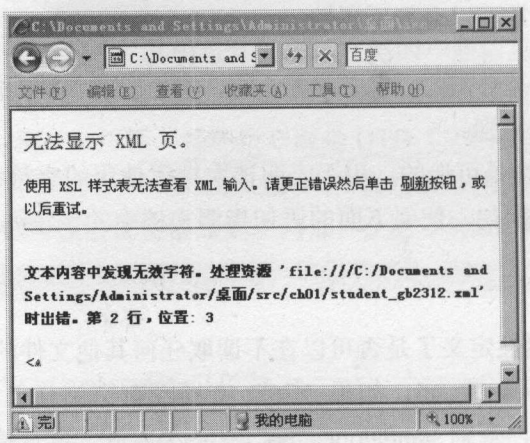


图 1-4　字符编码错误

错误提示的信息是“文本内容中发现无效字符”，这是因为在 XML 文档声明语句中没有明确指定文档中的字符编码方式，浏览器就会用默认的 Unicode 编码来解析该文档，而该文档中的字符实际上使用的是 GB2312 编码，而非 Unicode 编码。

注意　XML 支持多种字符集类型。但一般常用的编码有如下几种：简体中文码（GB2312）、繁体中文码（BIG5）、UTF-8。

另外，序言部分还可以包括下列可选组成部分。

- 文档类型声明（DTD），它定义了文档的类型和结构。
- 一个或多个处理指令，它提供了 XML 处理器传递给应用的信息。

1.2.2　文档元素

XML 文档的第二个主要部分是文档元素（根元素）。在 XML 文档中，元素指出了文档的逻辑结构，并且包含了文档的信息内容（在示例文档中是学生的信息，例如姓名、年龄和电话）。一个典型的元素有起始标签、元素内容和结束标签。元素的内容可以是字符数据、其他（嵌套的）元素或者两者的组合。

整个 XML 文档就是由标记和字符数据混合成的。标记是用来描述文档结构的定界文本，即元素的起始标签、元素的结束标签、空元素标签、注释、文档类型声明、处理指令、CDATA 节定界符、实体引用和字符引用等。

在示例文档中，文档元素是 students。其起始标签是<students>，结束标签是</students>，其内容是两个嵌套的 student 元素，每个 student 元素同样包括一系列的嵌套元素。

注意　XML 文档中的文档元素类似于 HTML 页中的<BODY>元素，但 XML 文档元素可以具有任意合法的名称。

1.3　XML 文档内容

XML 文档必须有一个根元素，用来包含可能有的其他内容。XML 文档中的所有内容都应该出现在根元素的内部。在遵守 XML 命名规则的前提下，用户可以为元素和属性选择任何名字。XML 文档内容的主体部分一般由根元素、子元素、属性、注释和内容组成。

XML 文档的内容结构如图 1-5 所示。

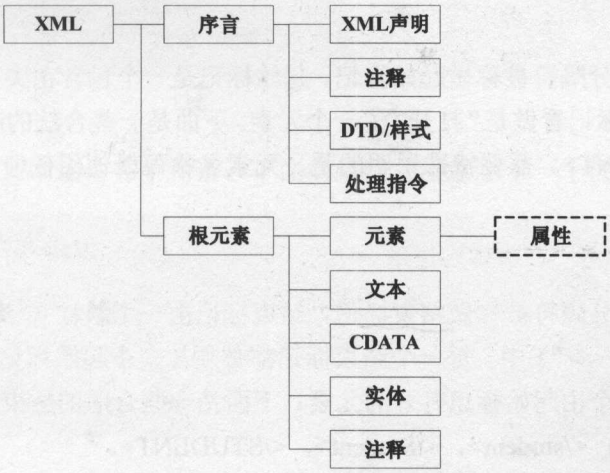


图 1-5　XML 内容结构图

1.3.1　XML 元素

元素是 XML 文档的基本组成部分。它们可以包含其他的元素、字符数据、字符引用、

实体引用、PI（Processing Instruction，处理指令）、注释或 CDATA 部分——这些合在一起被称做元素内容（element content）。所有的 XML 数据（除了注释、PI 和空白）都必须包容在元素中。

XML 文档中其他所有元素都是根元素的后代（子元素），student.xml 中的根元素是<students>，而<student>、<name>、<age>、<tel>都是它的子元素。

在 XML 文档中，元素有很多作用，它们可以标记内容；为它们标记的内容提供一些描述；为数据的顺序和相对重要性提供信息；展示数据之间的关系。

XML 中元素使用标记进行分隔，标记由一对尖括号（“<>”）围住元素名称（一个字符串）构成。每一个元素都必须由一个起始标记和一个结束标记分隔开。

在 XML 中，基本上没有什么保留字，可以用任何字符串来作为元素名称，但是 XML 元素的命名必须遵守下列规范。

- 元素的名字可以包含字母、数字和其他合法字符，且区分大小写。
- 元素的名字不能以数字或者标点符号开头。
- 元素的名字不能以 XML（或者 xml、Xml、xMl 等）开头。
- 元素的名字不能包含空格，并且避免使用“-”、“.”、“:”等特殊字符。
- 元素的命名应该遵循简单易读的原则。
- 如果 XML 文档与数据表对应，应尽量让 XML 文档中元素的命名和数据库中字段的命名保持一致，这样可以方便数据变换。
- 非英文字符、字符串也可以作为 XML 元素的名字，例如<姓名>、<年龄>等，但为了得到更好的支持，建议使用英文字母来进行命名。

1. 起始标记

一个元素开始的分隔符被称做起始标记。起始标记是一个包含在尖括号（“<>”）里的元素名称。可以把起始标记看做是“打开”了一个元素。下面是一些合法的起始标记：<student>，<Student>，<STUDENT>。需要特殊说明的是，元素名称可以使用任何合法字母，而不一定是 ASCII 码字符。

2. 结束标记

一个元素最后的分隔符被称做结束标记。结束标记由一个斜杠（“/”）和元素名称组成，被括在一对尖括号（“<>”）中。每一个结束标记都必须与一个起始标记相匹配，可以把结束标记理解为关闭了一个由起始标记打开的元素。下面是一些合法的结束标记，它们与前面列举的起始标记相对应：</student>，</Student>，</STUDENT>。

带有完整的起始、结束标记的元素应该是如下形式：

```
<自定义标记>包含的内容</自定义标记>
```

3. 元素分类

XML 文档中一共有四类元素：空元素，仅含文本的元素，仅含子元素的元素，含子元

素、文本或混合元素的元素。

（1）空元素

如果元素中不包含任何文本或子元素，那么它就是一个空元素。对于空元素，可以只加入起始标记和结束标记而不在其中包含任何内容，如下述代码所示：

```
<student></student>
```

空元素的精简表示方式是：由一个元素名称紧跟一个斜杠（“/”）组成，并括在一对尖括号（“<>”）中。下面的代码与上述代码的功能是一致的。

```
<student/>
```

（2）仅含文本的元素

有些元素含有文本内容。下述代码中<name>和<age>都是含有文本的元素：

```
<name>Rose</name>
<age>16</age>
```

（3）仅含子元素的元素

一个元素可以包含其他的元素。容器元素称为父（parent）元素，被包含的元素称为子（child）元素。例如<student>元素就是一个包含子元素的元素：

```
<student sex = "male">
    <name>Tom</name>
    <age>14</age>
    <tel>88889999</tel>
</student>
```

（4）混合元素

混合元素既含有文本也含有子元素。下面的代码片段显示了一个混合元素：

```
<student sex = "male">
大一四班
    <name>Tom</name>
    <age>14</age>
    <tel>88889999</tel>
</student>
```

4. 元素嵌套

XML 对元素有一个非常重要的要求——它们必须正确地嵌套。也就是说，如果一个元素（通过起始标签和结束标签来分隔）在另一个元素内部开始，那么也必须在同一个元素内部结束。例如，下列这些元素的是格式良好的：

```
<student>
    <name>Tom</name>
    <age>14</age>
    <tel>88889999</tel>
</student>
```

而下列这些元素的格式是不正确的：

```
<student>
    <name>Tom
    <age>14</age>
    <tel>88889999</tel>
</student></name>
```

当遇到没有被正确使用的嵌套标记时，XML 解析器会立刻报告一个“not well-formed（非格式良好的）”错误报告，在 IE 浏览器中的错误信息如图 1-6 所示。

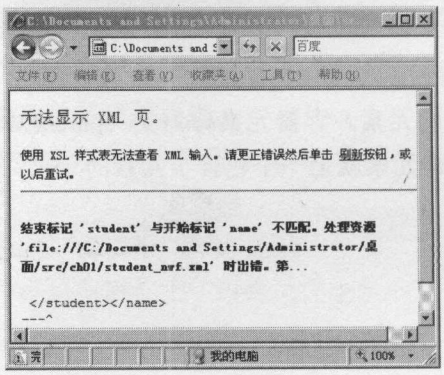


图 1-6　元素嵌套错误

综上所述，在 XML 文档中使用元素时应注意以下几点要求。

- 元素必须含有起始标签和结束标签。
- 在没有内容（空元素）的情况下，建议使用省略写法。
- 标签名称必须符合 XML 命名规则。
- 元素必须正确地嵌套。

1.3.2　XML 属性

XML 元素可以拥有属性。属性是对标记进一步的描述和说明，一个标记可以有多个属性。XML 中的属性与 HTML 中的属性是一样的，每个属性都有它自己的名字和数值，属性是标记的一部分。例如：

```
<student sex = "male">
</student>
```

XML 中的属性也是由用户自己定义的，属性由“名称/值”对组成，其中的值是包含于单引号或双引号中的；一个元素可以有多个属性，它的基本格式为：

```
<元素名 属性名="属性值">
```

特定的属性名称在同一个元素标记中只能出现一次；属性值不能包括“<”、“>”、“&”。

例如 student 元素中可以使用属性表示学生的性别：

```
<student sex = "male"></student>
```

也可以这样写：

```
<student sex = 'male'></student>
```

上面的两种写法在一般情况下是没有区别的，双引号的写法更普遍一些。

当元素包含属性时，常称为复合类型（complex type）元素，在书写 XML 模式文档时是常见的应用模式。

在 XML 中，可以将属性改写为嵌套的子元素。例如，对于下列代码：

```
<student sex = "female">
    <name>Rose</name>
    <age>16</age>
    <tel>66667777</tel>
</student>
```

可以改写为：

```
<student>
    <sex>female</sex>
    <name>Rose</name>
    <age>16</age>
    <tel>66667777</tel>
</student>
```

上述两种写法都能够正确地描述数据，哪种写法更好并没有一个明确的规则，两种写法都是可接受的。

属性在 HTML 中可能十分便利，但在 XML 中，最好避免使用属性。因为使用属性时会引发以下一些问题。

- 属性不能包含多个值（而子元素可以）。
- 属性不容易扩展。
- 属性不能够描述结构（而子元素可以）。
- 在使用程序代码进行处理时，属性比子元素要难解析。

- 属性值很难通过 DTD 进行测试。
- 使用属性来存储数据，XML 文档比较难以阅读和操作。

注意　在 XML 定义中经常用到属性，但在描述数据时建议尽量使用元素，仅在描述那些与数据关系不大的额外信息时使用属性。

1.3.3　注释

XML 中，注释以“<!--”开始，以“-->”结束，除了在 XML 声明之前，注释可以出现在 XML 文档的其他任何位置。在进行 XML 解析时，注释内的任何标记都被忽略，如果希望除去 XML 文档的一块较大部分，只需用注释标记括住那个部分即可。

XML 中，注释的语法如下所示。

```
<!-- 这里是注释信息 -->
```

例如：

```
<!-- File Name:student.xml -->
```

加入注释通常是为了便于阅读和理解，注释并不影响 XML 文档的处理。在添加注释时需要遵循以下规则。

- 注释里不能包含文本“--”。
- 注释不能包含于标记内部。
- 元素中的开始标签或结束标签不能被单独注释掉。

1.3.4　字符引用和实体引用

与 SGML 和 HTML 一样，XML 为显示非 ASCII 码字符集中的字符提供了两种方法：字符引用和实体引用。

1. 字符引用

实际处理过程中，不在键盘上的字符或是图形字符是无法直接输入的，这种情况下，可以使用 Unicode 码将它们以字符引用的形式加入。比如，可以将版权符号“©”编码成“©”。以“&#”开始并以“;”结束的引用都是字符引用。中间的数字是所需字符的 Unicode 编码，如果编码写成十六进制形式，应该使用一个“x”作为前缀，比如“©”也可表示为“©”。下述代码是两个示例：

```
&#AAAAA;
&#XXXXX;
```

上面的字符串“AAAAA”和“XXXXX”可能是一个或多个数字，对应着任何 XML 允许的 Unicode 字符值，可以是十进制数字或十六进制数字。

下述代码用于实现任务【描述 1.D.3】，演示了如何使用字符引用。

【描述 1.D.3】 chars.xml

```
<?xml version="1.0" ?>
<chars>
    <ch>&#169;</ch>
    <ch>&#xA9;</ch>
    <ch>&#174;</ch>
    <ch>&#xAE;</ch>
</chars>
```

通过 IE 浏览器查看结果如图 1-7 所示。

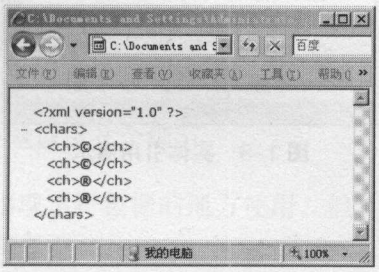


图 1-7　字符引用演示

2. 实体引用

实体引用允许在元素内容或属性值中插入任何字符串，这就为字符引用提供了一种助记的替代方式。实体引用方式是在一个合法的 XML 名字前面加上一个“&”符号，后面加上一个分号“;”，如下所示：

```
&name;
```

有五个实体被定义为 XML 的固有部分，它们通常作为 XML 标记分隔符号的转义序列，如表 1-2 所示。

表 1-2　XML 的转义字符

实体	用途
<	通常用来替换小于号（<）
>	通常用来替换大于号（>）
&	通常用来替换字符（&）
"	通常用来替换双引号（"）
'	通常用来替换单引号（'）

下述代码用于实现任务【描述 1.D.4】，演示了如何使用实体引用。

【描述 1.D.4】 chars2.xml

```
<?xml version="1.0" ?>
```

```
<chars>
    <ch>&lt;</ch>
    <ch>&gt;</ch>
    <ch>&amp;</ch>
    <ch>&quot;</ch>
    <ch>&apos;</ch>
</chars>
```

通过 IE 浏览器查看结果，如图 1-8 所示。

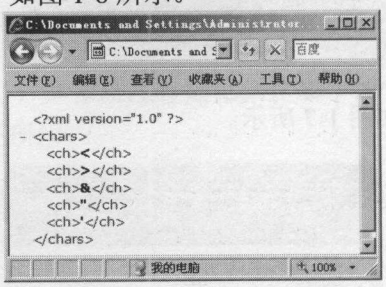


图 1-8　实体引用演示

1.3.5　CDATA 节

一般情况下，为了能在元素内容的字符数据中插入特殊字符（如<、>、&等），可使用字符引用或者一个预定义的通用实体引用。但如果存在大量的特殊字符，使用这种方式就很笨拙而且使数据难以阅读。在这种情况下，可以将包含限制字符的文本放在 CDATA 节中。

CDATA 节以“<![CDATA[”开始，并以“]]>”结束。在这两个限定字符组之间，可以输入除了“]]>”之外（因为它会被解释为 CDATA 节的结束）的任意字符。CDATA 节中的所有字符都会被当做元素字符数据的常量部分，而不是 XML 标记。

下述代码用于实现任务【描述 1.D.5】，演示了 CDATA 节的使用。

【描述 1.D.5】 CDATA.xml

```
<?xml version="1.0" encoding="GB2312"?>
<files>
    <file>
        <name>special.txt</name>
        <!--使用 CDATA 节可以输入除]]外的任何字符 -->
        <content><![CDATA[some special charactor " ' < > & ]]></content>
    </file>
    <file>
        <name>special.txt</name>
        <!--不使用 CDATA 节需要使用字符引用或实体引用来输入特殊字符 -->
        <content>some special charactor &quot; &apos; &lt; &gt; &amp; </content>
    </file>
```

```
</files>
```

通过 IE 浏览器查看结果如图 1-9 所示。

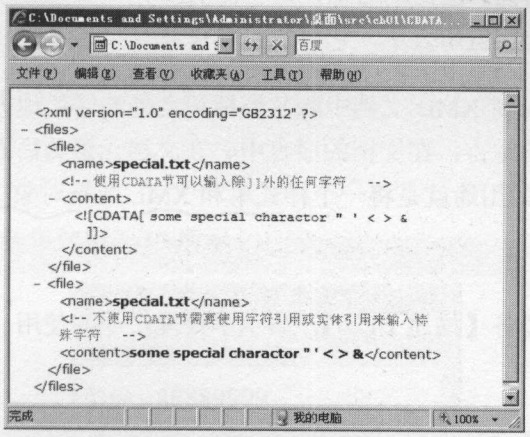


图 1-9　CDATA 演示

在使用 CDATA 节时，应注意以下几个细节。

- CDATA 节可以在任何出现字符数据的地方使用，但不能出现在 XML 标记中。
- CDATA 节必须包含在元素中。
- CDATA 节之间不能再包含 CDATA 节（不能嵌套）。

如下示例是格式不正确的。

```
<?xml version="1.0" encoding="GB2312"?>
<files>
    <file>
        <!--包含在标记中 -->
        <name <![CDATA[some description]]>>special.txt</name>
        <!--CDATA 嵌套 -->
        <content><![CDATA[outer <![CDATA[inner]]> ]]></content>
    </file>
</files>
```

1.3.6　处理指令

XML 并不预先假设元素或者其内容的处理办法，这是 XML 的优势之处。在实际开发过程中，经常需要把某些信息通过文档传递给应用程序，处理指令（Processing Instruction，PI）正是 XML 为此目的提供的一种机制。

处理指令的语法形式如下：

```
<? target instruction ?>
```

具体介绍如下。

- target：指令所指向的应用的名称，是必需的部分，而且必须是有效的 XML 名称。
- instruction：一个字符串表示，它可能包含任何有效的字符（除了“?>”）。

可以把处理指令插入到 XML 文档中除其他标记之外的任何地方，也就是说，可以把它插入到与插入注释相同的地方：在文档的序言中、在文档元素的后面或者在元素的内容中。一个几乎随处可见的 PI 的用途就是将一个样式单和 XML 数据对象关联起来：

```
<?xml-stylesheet ... ?>
```

下述代码用于实现任务【描述 1.D.6】，演示了处理指令的使用。

【描述 1.D.6】 PI.xml

```
<?xml version="1.0" encoding="GB2312"?>
<!-- 序言中的处理指令 -->
<?xml-stylesheet type="text/css" href="student.css" ?>
<students>
    <!-- 元素内部的处理指令 -->
    <?ScriptA level="A" ?>
    <student sex = "male">
        <name>Tom</name>
        <age>14</age>
        <tel>88889999</tel>
    </student>
    <student sex = 'female'>
        <name>Rose</name>
        <age>16</age>
        <tel>66667777</tel>
    </student>
</students>
```

该 XML 文档关联了一个 student.css 文件，用于设置 XML 文档中元素的样式。代码如下所示。

【描述 1.D.6】 student.css

```
student{
    display:block;
    margin-top:12pt;
    font-size:10pt
}
name{
    font-style:italic;
```

```
    font-size:20pt
}
age{
    font-weight:bold
}
tel{
    font-size:20pt
}
```

通过 IE 浏览器查看结果如图 1-10 所示。

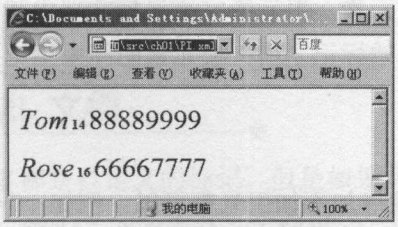


图 1-10　处理指令演示

在 XML 中使用 PI 有如下好处。

- 可以作为脚本或服务端包含文件的挂钩（避免类似 HTML 语法中“<! -->”注释语法的泛滥成灾）。
- 可以作为扩展模式的机制（否则它们就不能被修改）。
- 它是一种无须改变 DTD 认证就可以扩展文档的方法。
- 在不影响文档结构的情况下，通过在 XML 文件中嵌入处理指令，将信息以文档的形式传递给应用程序。

注意　处理指令不能放入 XML 标记中声明。

1.4　XML 文档规则

XML 语法虽然简单，但要遵循“良好格式”的规则才能编写合法的 XML 应用。

1.4.1　格式良好的 XML 文档规则

1. 必须有声明语句

XML 声明是 XML 文档的第一句，其格式如下：

```
<?xml version="1.0" standalone="yes/no" encoding="UTF-8"?>
```

XML 声明的作用是告诉浏览器或者其他处理程序：这个文档是 XML 文档。

2. 注意大小写

在 XML 文档中，大小写是有区别的。“<P>”和“<p>”是不同的标记。注意在写元素时，前后标记大小写要保持一致。例如：<Author>TOM</Author>，写成<Author>TOM</author>是错误的。

最好养成一种习惯，在输入标记时或者全部大写，或者全部小写，或者第一个字母大写，这样可以减少因为大小写不匹配而产生的文档错误。

3. XML 文档有且只有一个根元素

良好格式的 XML 文档必须有一个根元素，也就是紧接着声明后面建立的第一个元素，其他元素都是这个根元素的子元素，根元素完全包括文档中其他所有的元素。根元素的起始标记要放在其他所有元素的起始标记之前；根元素的结束标记要放在其他所有元素的结束标记之后。

4. 属性值使用引号

在 HTML 代码里面，属性值加不加引号都可以。例如：“word”和“word”都可以被浏览器正确解释。但是 XML 中规定所有属性值必须加引号（可以是单引号，也可以是双引号，建议使用双引号），否则将被视为错误。

5. 所有的标记必须有相应的结束标记

在 HTML 中，标记可以不成对出现，而在 XML 中，所有标记必须成对出现，有一个起始标记，就必须有一个结束标记，否则将被视为错误。

6. 所有的空标记也必须被关闭

空标记就是标记对之间没有内容的标记。比如“<img>”等标记。在 XML 中，规定所有的标记必须有结束标记。

7. 标记必须正确嵌套

标记之间不得交叉。在 HTML 文件中，可以这样写：

```
<B><H>Today is Saturday.</B></H>
```

<B>和<H>标记之间有相互重叠的区域，而在 XML 中是严格禁止这样标记交错的写法，标记必须以规则性的次序出现。

8. 处理特殊字符

在 XML 文档中，如果要用到特殊字符，必须用相应符号代替，如“<”已作为标签使用，不能出现在 XML 文档中，应以相应的实体引用代替。

对于空白字符，XML 的处理方式和 HTML 不一样。HTML 标准规定，不管有多少个空格，都当做一个空格来处理。例如在 HTML 中，“Today is Saturday.”将会被显示成“Today is Saturday.”，HTML 解析器会自动把句子中多余的空格部分去掉。

而在 XML 中规定，所有标记以外的空格，解析器都要全部交给应用程序处理，即解析

器会保留内容中所有的空格字符并不加修改地传递给应用程序。

1.4.2　格式良好的 XML 文档

一个遵守 XML 语法规则，并遵守 XML 规范的文档称之为格式良好的 XML（Well-formed XML）。XML 必须是格式良好的，才能够被解析器正确地解析出来。一个格式良好的 XML 文档需满足下列条件。

- 语法合乎 XML 规范。
- 元素构成一个层次树，只有一个根结点。
- 除非提供了 DTD，否则没有对外部实体的引用。

1.4.3　有效的 XML 文档

在 XML 文档中，用的大多都是自定义的标记。但是如果两个同行业的公司 A 和 B 要用 XML 文档相互交换数据的话，他们之间必须有一个约定——即可以用哪些标记，父元素中能够包括哪些子元素，各个元素出现的顺序，元素中的属性怎样定义等，这样双方在用 XML 交换数据时才能够畅通无阻。这种约定规则可以用 DTD（Document Type Definition，文档格式定义）或 XML Schema（XML 模式）来表述。

注意　XML Schema 是基于 XML 的 DTD 的替代品，W3C 使得 DTD 和 Schema 可以相互替代。关于 DTD 和 Schema，将在后续章节详细介绍。

一个有效的 XML 文档首先应该是一个格式良好的 XML 文档，其次还必须符合 DTD 或是 XML 模式所定义的规则。所以说格式良好的 XML 文档不一定是有效的 XML 文档，但有效的 XML 文档一定是格式良好的 XML 文档。

下述代码用于实现任务【描述 1.D.7】，基于 DTD，演示有效 XML 文档的定义。

【描述 1.D.7】 product.dtd

```
<?xml version="1.0" encoding="GB2312"?>
<!ELEMENT PRODUCTS (PRODUCT)+>
<!ELEMENT PRODUCT (PRODUCTNAME,DESCRIPTION,PRICE,QUANTITY)>
<!ELEMENT PRODUCTNAME (#PCDATA)>
<!ELEMENT DESCRIPTION (#PCDATA)>
<!ELEMENT PRICE (#PCDATA)>
<!ELEMENT QUANTITY (#PCDATA)>
<!ATTLIST PRODUCT PRODUCTID ID #REQUIRED CATEGORY (BOOKS|TOYS) "TOYS">
```

下面的 XML 是一个应用上述 DTD 文档的有效的 XML 文档。

【描述 1.D.7】 product.xml

```
<?xml version="1.0" encoding="GB2312"?>
<!DOCTYPE PRODUCTDATA SYSTEM "product.dtd">
<PRODUCTS>
    <PRODUCT PRODUCTID="P001" CATEGORY="TOYS">
        <PRODUCTNAME>乱世佳人</PRODUCTNAME>
        <DESCRIPTION>以美国内战为背景进行叙事</DESCRIPTION>
        <PRICE>26.80</PRICE>
        <QUANTITY>60</QUANTITY>
    </PRODUCT>
</PRODUCTS>
```

通过 IE 浏览器查看结果如图 1-11 所示。

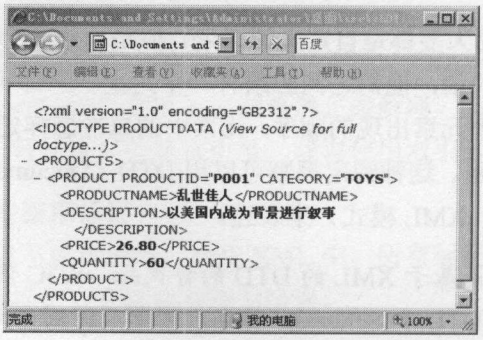


图 1-11 DTD 演示

DTD 定义了 XML 文档中可用的合法元素。它通过定义一系列合法的元素决定了 XML 文档的内部结构。

小结

通过本章的学习，学生应该能够学会：

- XML 是一种类似于 HTML 的标记语言，是一种元标记语言。
- XML 是用来描述数据的，不是 HTML 的替代品。
- XML 的标记不是在 XML 中预定义的，用户可以自定义标记。
- XML 有两个先驱：SGML 和 HTML，XML 正是为了解决它们的不足而诞生的，XML 是一个精简的 SGML 子集。
- XML 文档有两个主要组成部分：序言和根元素。
- XML 文档内容的主体部分一般由根元素、子元素、属性、注释和内容组成。
- 元素是 XML 文档的基本组成部分。它们可以包含其他的元素、字符数据、字符引用、实体引用、PI、注释或 CDATA 部分。

- 字符引用和实体引用是 XML 处理特殊字符的两种方式。
- 引入 CDATA 节可以描述除了“]]>”之外的任意字符串。
- 处理指令可以把某些信息以文档的形式传递给应用程序。
- XML 必须是格式良好的，才能够被解析器正确地解析。
- 一个有效的 XML 文档应该既是格式良好的，同时还必须是符合 DTD 或是 XML Schema 所定义规则的 XML 文档。

练习

1. 属性_____用来表示 XML 文档所使用的字符集。

   A. version　　B. encoding　　C. standalone　　D. language

2. 含有简体中文字符的 XML 文档中，encoding 属性值可设为_____。（多选）

   A. UTF-8　　B. BIG5　　C. GB2312　　D. ISO-8859-1

3. 实体引用前面带有一个符号_____。

   A. &　　B. #　　C. +　　D. ;

4. 下述关于 XML 文档的描述正确的是_____。（多选）

   A. XML 不区分大小写

   B. 任何 XML 文档有且只有一个根元素

   C. XML 中的标记可以没有结束标记

   D. 在合适的位置，引入 CDATA 节可以描述除了“]]>”之外的任意字符串

5. 每个 XML 文档都分为两个部分：_____和_____

   <?xml version="1.0"　encoding="gb2312"?>是一个_____

6. XML 中一共有四类元素，分别是_____、_____、_____和_____。

7. XML 文档内容的主体部分，一般由_____、_____、_____、注释和内容组成。

8. XML 是从_____扩展来的。

9. 使用 XML 的优点是什么？

10. 相对于 EDI 的结构化信息技术而言，XML 的优势有哪些？

11. 简要描述 XML 的命名规范。

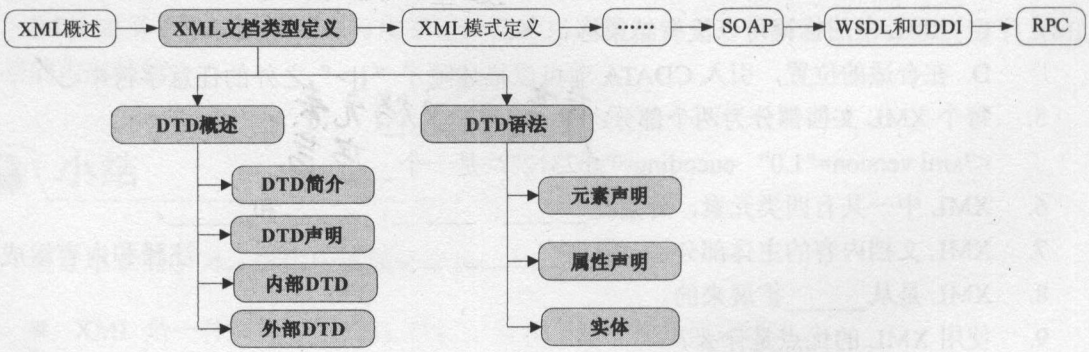
12. 简要描述格式良好的 XML 文档应遵循的规则。

第 2 章　XML 文档类型定义

本章目标

- 了解 DTD 的作用
- 掌握内部 DTD 的使用
- 掌握外部 DTD 的使用
- 掌握 DTD 的文档结构
- 掌握 DTD 中不同类型元素的定义及使用
- 掌握 DTD 中各种属性的定义及使用
- 掌握 DTD 中实体的定义及使用

学习导航



任务描述

【描述 2.D.1】

创建并使用内部 DTD 规范 XML 文档。

【描述 2.D.2】

创建并使用外部 DTD 规范 XML 文档。

【描述 2.D.3】

创建并使用带属性的 DTD，演示 CDATA 的使用。

【描述 2.D.4】

创建并使用带 NMTOKEN 属性的 DTD，演示 NMTOKEN 的使用。

【描述 2.D.5】

创建并使用带 ID 属性的 DTD，演示 ID 的使用。

【描述 2.D.6】

创建并使用带 IDREF（IDREFS）属性的 DTD，演示 IDREF 的使用。

【描述 2.D.7】

创建并使用带枚举类型属性的内部 DTD，演示枚举类型的使用。

【描述 2.D.8】

创建并使用带自定义实体的内部 DTD，演示自定义实体的使用。

2.1 DTD 概述

XML 的精髓在于“无障碍”数据交换和数据共享，编写格式良好的 XML 文档，使其他用户能够理解符合我们创建的 XML 词汇表（包括元素及属性等）的文档结构，须通过某种通用的方式说明词汇表的语法规则。为此，XML1.0 提供了一种机制——文档类型定义（Document Type Definition，DTD），并将其作为规范的一部分。DTD 使用正式的语法定义 XML 文档的结构和允许值。

2.1.1 DTD 简介

XML 的最大灵活性在于允许用户制定基于信息描述、体现数据之间逻辑关系的自定义标记，确保文档具有较强的易读性、清晰的语义和易检索性。因此，一个完全意义上的 XML 文档不仅仅是“Well Formed”（格式良好的），而且还应该是使用了一些自定义标记的“Validating XML”（有效的）文档，即它必须遵守文档类型定义 DTD 中已声明的各种规定。文档类型定义 DTD，用来描述 XML 文档的结构。DTD 定义了 XML 文档中可用的合法元素，并且定义了可以在文档中存在的元素、元素可以具有的属性、元素的层次结构，以及元素在整个文档中出现的顺序。简而言之，DTD 规定了一个语法分析器以解释一个“Validating XML”文档所需要知道的所有规则的细节。

DTD 可以看做是一个或多个 XML 文件的模板，这些 XML 文件中的元素、元素的属性、元素的排列方式/顺序、元素能够包含的内容等，都必须符合 DTD 中的定义。XML 文件中的元素（标记）是根据应用的实际情况来创建的。想要创建一份完整性高、适应性广的 DTD 是非常困难的，因为各行各业都有他们自己的行业特点，所以 DTD 通常是以某种应用领域为定义的范围，如：生产、医学、工商、金融。DTD 定义的元素涵盖范围越广泛，它就越复杂。

引入 DTD 的优势如下所示。

- 引入 DTD，每个 XML 文件可以携带一个自身格式的描述。
- 引入 DTD，不同组织的人可以使用一个通用 DTD 交换数据，应用程序可以使用一个标准 DTD 校验从外部世界接收来的 XML 数据是否有效。
- 引入 DTD，便于在网络上进行数据的共享和交互。例如，两个相同行业不同地区的人使用同一个 DTD 文件作为文档的创建规范，那么他们的数据就很容易交换和共享。网络上其他用户补充的数据，只需要根据公用的 DTD 规范来建立即可。

目前，针对不同的行业和应用，已经有数量众多的写好的 DTD 文件可以利用，这些 DTD 文件已经建立了通用的元素和标签规则。用户无须重新创建，只要在这些 DTD 文件的基础上加入特定的新元素即可。当然，用户也可以创建自己的 DTD。

2.1.2　DTD 声明

DTD 是与 XML 文档相关的。通常，XML 文档中通过 DOCTYPE 声明将 XML 与 DTD 建立关联的指令，当验证有效性的解析器读到该指令时，它获取相应的 DTD，并根据其中定义的规则对文档进行校验。

DOCTYPE 声明由以下部分组成：关键字、文档的根元素名称、可选的外部标记符，以及可选的标记声明块。外部标记符用于外部 DTD（外部子集）的命名和定位，标记声明块是由标记声明（内部子集）构成的。

DOCTYPE 声明语法示例如下：

```
<?xml version="1.0" standalone="no" encode="UTF-8"?>
<!DOCTYPE  Catalog ...>
<Catalog >
```

第 2 行中使用“<!DOCTYPE>”标记，该标记就是 DOCTYPE 声明，说明该文档的第一个元素（根元素）是 Catalog，省略号“…”隐藏了 DOCTYPE 声明的其余部分。

XML 规范定义了两种 DOCTYPE 声明的方法。

- 内部 DTD 声明：在 DOCTYPE 声明体中直接定义内部标记子集。
- 外部 DTD 声明：用户可以在独立的 DTD 文件中提供外部标记子集声明，再通过 DOCTYPE 声明将其引入。

这两种声明方式可以同时使用。

注意　DOCTYPE 声明必须位于 XML 声明之后，且在任何文档元素之前。但是，XML 声明和 DOCTYPE 声明之间可以插入注释和处理指令。

2.1.3　内部 DTD

可以使用“<!DOCTYPE[.....]>”语句在 XML 文档序言部分声明 DTD 文档的定义，引入内部 DTD 的 XML 文档的语法结构如下：

```
<?xml version = "1.0" encoding="GB2312"?>
<!DOCTYPE element-name[
元素描述
]>
<!--文档数据区..... -->
```

其中：

- <!DOCTYPE：表示开始设定 DTD，注意 DOCTYPE 必须大写。
- element-name：指定此 DTD 的根元素的名称，一个 XML 文件只能有一个根元素。

注意，如果 XML 文档使用了 DTD，那么文件中的根元素就在这里指定。

- [ ]>：在[ ]标记里面定义 XML 文件中使用的元素，然后用“>”结束 DTD 的定义。

下述代码用于实现任务【描述 2.D.1】，创建并使用内部 DTD 规范 XML 文档。

【描述 2.D.1】 student_1.xml

```
1.  <?xml version="1.0" encoding="GB2312"?>
2.  <!DOCTYPE students [
3.  <!ELEMENT students (student)+>
4.  <!ELEMENT student (name,age,tel)>
5.  <!ELEMENT name (#PCDATA)>
6.  <!ELEMENT age (#PCDATA)>
7.  <!ELEMENT tel (#PCDATA)>
8.  ]>
9.  <students>
10.     <student>
11.         <name>Tom</name>
12.         <age>14</age>
13.         <tel>88889999</tel>
14.     </student>
15. </students>
```

上述文档是一个格式良好，并且有效（满足 DTD 校验）的 XML 文档，其中：

- 第 1 行代码是一个 XML 声明，表示文档遵循的是 XML 的 1.0 版的规范。
- 第 2～8 行即为内部 DTD 的定义内容。这个内部 DTD 中，定义了名为 students 的元素是 XML 文档中的根元素，此根元素拥有至少一个 student 子元素，子元素 student 有 name、age 和 tel 三个子元素。name、age 和 tel 元素都为“#PCDATA”类型。

注意 在进行 DTD 书写时，特别注意元素定义时，需要在元素名称和内容之间加空格隔开，如：“<!ELEMENT students”和“(student)+>”之间要通过空格隔开。

2.1.4 外部 DTD

外部 DTD 是一个独立于 XML 文档的文件，实际上也是一个文本文件，只是使用.dtd 作为文件扩展名。外部 DTD 独立于 XML 文档，所以它可以供多个 XML 文件使用，就像用同一个模板可以写出多个不同内容的文件一样。

使用外部 DTD 的好处是：它可以方便高效地被多个 XML 文档所共享。只要写一个 DTD 文档，就可以被多个 XML 文档所引用。

除了没有内部 DTD 中的<!DOCTYPE [.....]>语句外，外部 DTD 的创建方式、语法和内部 DTD 是一样的。XML 文档使用 DTD 文件的声明语句来引用创建好的外部 DTD 文件，此语

句必须位于 XML 文档的文件序言区，并且要紧跟在 XML 声明语句后面，DTD 文件的声明语句格式如下：

```
<!DOCTYPE type-of-doc SYSTEM/PUBLIC "dtd-name">
```

具体介绍如下所示。

- <!DOCTYPE：是指要定义一个 DOCTYPE。
- type-of-doc：指定文档类型的名称，由用户自己定义，通常与使用这个 DTD 文件的 XML 文档的根元素名称一致。
- SYSTEM/PUBLIC：这两个参数只用其一。SYSTEM 是指文档使用的是私有的外部 DTD 文件，这个关键字主要用于引用一个作者或组织所编写的众多 XML 文档中通用的 DTD；而 PUBLIC 则指文档调用一个公用的外部 DTD 文件，此外部 DTD 文件经过了公开讨论，即一个由权威机构制定的、提供给特定行业或公众使用的 DTD。
- dtd-name：是存放 DTD 文件的地址和名称。

综上所述，可以这样理解引入外部 DTD 的 XML 文档的结构，即：

```
< ?xml version = "1.0"?>
<!DOCTYPE 根元素名 SYSTEM/PUBLIC "外部 DTD 文件名及其位置">
<!--文档数据区..... -->
```

下述代码用于实现任务【描述 2.D.2】，创建并使用外部 DTD 规范 XML 文档。

定义外部 DTD 文档内容如下所示。

【描述 2.D.2】 student_2.dtd

```
<?xml version="1.0" encoding="GB2312"?>
<!ELEMENT students (student+)>
<!ELEMENT student (name,age,tel*)>
<!ELEMENT name (#PCDATA)>
<!ELEMENT age (#PCDATA)>
<!ELEMENT tel (#PCDATA)>
```

关联外部 DTD 文件的 XML 的总体代码如下：

【描述 2.D.2】 student_2.xml

```
<?xml version="1.0" encoding="GB2312"?>
<!DOCTYPE students SYSTEM "student_2.dtd">
<students>
    <student>
        <name>Tom</name>
        <age>14</age>
```

```
        <tel>88889999</tel>
    </student>
</students>
```

上述代码中，第 2 行代码的作用就是将外部 DTD 文件 student_2.dtd 应用于此 XML 文档。

2.2 DTD 语法

DTD 是保证 XML 文档有效性的方法之一，可使 XML 解析器通过比较 XML 文档和 DTD 文件来确定文档是否符合规范、元素和标签使用是否正确等。一个 DTD 文档包含：元素的定义规则、元素间关系的定义规则、元素可使用的属性以及可使用的实体或符号规则等。DTD 文件也是一个 ASCII 的文本文件，后缀名为.dtd。

2.2.1 元素声明

元素是 XML 的核心与灵魂，它包含了实际的文档信息，并指出了这些信息的逻辑结构。元素以树形分层机构排列，可以嵌套在其他元素中。

在 DTD 中，元素类型是通过 ELEMENT 标记声明的。除了关键字，标记还提供所声明类型的名称和内容规范。正如第 1 章所述，元素类型名要遵守 XML 对名称的限制。名称可以是字母、数字，也可以使用标点符号，如：冒号（:）、下画线（_）、连字符（-）和句点（.）。然而，名称不能以数字开头。它的第一个字符只能是字母、下画线或冒号。

注意 虽然名称中可以使用冒号，但是鉴于后面即将介绍的命名空间，最好避免在元素名称中使用冒号。

DTD 中使用元素类型声明 ETD（Element Type Declaration）来声明所有有效的文档元素。其语法结构如下：

```
<!ELEMENT element-name element-definition>
```

具体介绍如下所示。

- <!ELEMENT：表示开始元素设置，注意此处 ELEMENT 关键字必须是大写。
- element-name：表示要设置的元素的名称。
- element-definition：指明要对此元素进行怎样的定义，就是说<元素>...</元素>之间能够包含什么内容，是其他元素还是一般性的文字。

XML 中按元素的内容（element-definition）可以将元素划分为五类，不同类型的元素使用 DTD 进行定义时的语法如下。

1. ANY 类型

如果不需要对元素的内容进行限制，可以在 DTD 中使用 ANY 元素类型。定义方式为：

```
<!ELEMENT element-name ANY>
```

XML 文档里该元素中可以包含任何在 DTD 中定义的元素内容。建议一般只把文档的根元素规定为 ANY 类型。将根元素设为 ANY 类型后，元素出现的次数和顺序不受限制。

注意　为了增强可读性，每个元素定义通常占据单独的一行。

2. EMPTY 类型

EMPTY 元素类型。定义方式为：

```
<!ELEMENT element-name EMPTY>
```

这类元素在 XML 文档中使用空元素标记，即元素中没有内容，不能包含子元素和文本，但可以有属性。例如，对于下面的 DTD 定义：

```
<!ELEMENT student EMPTY>
```

以下这个元素是合法的：

```
<student name="Jack" age="16"/>
```

而下面这些元素的定义就是非法的，因为此元素应该为空元素，既不能有内容也不能有子元素：

```
<student>Jack</student>
<student><name>Jack</name></student>
```

3. #PCDATA 类型

#PCDATA 类型的元素（纯文本元素，或称简单元素）可以包含任何字符数据，但是不能在其中包含任何子元素。PCDATA 代表字符数据，为防止与关键字混淆，需加“#”前缀。其定义方式为：

```
<!ELEMENT element-name (#PCDATA)>
```

下面的 DTD 定义说明 name 只能包含字符数据，即非标记文本，并且它不能包含自己的子元素：

```
<!ELEMENT name (#PCDATA)>
```

下面这些元素定义都是合法的：

```
<name>Jack</name>
<name>2008</name>
<name>123.456</name>
```

因为 XML 不会去检查 PCDATA 的内容，只要是文本就可以。但下面的元素定义是非法的：

```
<name>
    <firstname>White</firstname>
    <lastname>Smith</lastname>
</name>
```

4. 父元素类型

父元素类型只能包含子元素，并且这些子元素外没有文本。这类元素在 DTD 中通过正则表达式规定子元素出现的顺序和次数。语法分析器将这些正则表达式与 XML 文档内部的数据模式相匹配，从而判断该文档是否是有效的 XML。表 2-1 列出了在 DTD 中用到的正则表达式的元组运算符。

表 2-1　正则表达式的元组运算符

符号	意义
?	使用“?”修饰的元素可以不出现或只出现一次
*	使用“*”修饰的元素可以不出现或出现多次
+	使用“+”修饰的元素必须至少出现一次以上
无符号	没有修饰符的元素只能出现一次
()	用来给元素分组
|	只能在列出的对象中选择一个
,	列出的对象必须按指定的顺序出现

表 2-2 通过示例介绍了上述元组运算符的用法。

表 2-2　用法示例

示例	说明
(A|B|C)	只能从 A、B、C 中选择一个出现
(A+)	A 必须出现，而且可以出现多次
(B*)	B 可以不出现，如果出现，可以出现多次
(C?)	C 最多可以出现一次，但可以不出现
(A,B,C)	A、B、C 必须出现，而且必须按顺序出现
(A|B|C),(D|E),F	分成三组

对于下述 DTD 定义：

```
<!ELEMENT students (student+)>
<!ELEMENT student (name,age,tel*)>
```

表明：

students 元素是父元素，它可以有至少一个 student 子元素（由“+”表示）；student 元素必须有一个 name 元素、一个 age 元素、零个或多个 tel 元素（由“*”表示），而且 name、age、tel 出现的次序必须一致。

5. 混合元素类型

混合元素类型是包含子元素和文本数据的混合体，混合元素的定义方式为：

```
<!ELEMENT 元素名 (#PCDATA | element-name1 | element-name2 | ……)*>
```

对于下述 DTD 定义就是一个混合元素：

```
<!ELEMENT students (#PCDATA|student)*>
```

在 DTD 定义中，带有字符数据的混合元素只能指定可出现的子元素的名称，而不能限定它们出现的顺序及每个元素的出现次数，或者它们是否出现，严重地限制文档的结构。因此，最好避免使用混合元素。

注意　混合元素在声明时，其语法结构中除了可以改变子元素数目以外，其他的任何改动都是不合法的。也就是说，不能在包括“#PCDATA”的元素声明中使用逗号、问号或加号；用竖线分隔的元素和“#PCDATA”的列表是合法的。其他用法是不合法的。

2.2.2　属性声明

属性是对元素的补充和修饰，它能够将一些简单的特性与元素相关联。通过属性，可以给元素绑定大量信息。例如，在 HTML 标记 IMG 中，SRC 就是一个属性。属性在 DTD 中是使用 ATTLIST 标记声明的。对于含属性的元素，至少要通过一个 ATTLIST 标记声明其属性列表。ATTLIST 声明由以下部分构成：ATTLIST 关键字、属性修饰的元素名称，以及零个或多个属性定义。属性的定义方式为：

```
<!ATTLIST 元素名 属性名称 属性类型 取值方式 >
```

实际运用中，一个元素往往不只有一个属性，当需要为一个元素定义多个属性时，也可采用如下定义方式：

```
<!ATTLIST 元素名
    属性名称 属性类型 取值方式
    属性名称 属性类型 取值方式
    ...
 >
```

其中，属性取值方式如表 2-3 所示。

表 2-3　属性取值方式

取值	含义
#REQUIRED	元素的每个实例必须包含该属性
#IMPLIED	元素的实例可以选择性地包含该属性

（续表）

取值	含义
#FIXED value	固定取值的属性，该属性值不允许被其他值替代。需要为一个特定的属性提供一个默认值，如果元素中不包含该属性，解析器将默认值作为属性值
默认值	事先定义了默认值的属性。需要在 DTD 中提供一个默认值，在 XML 文档中可以为该属性给出新的属性值，也可以不另外给出属性值，采用 DTD 中给出的默认值

属性类型用来指定该属性的取值类型，DTD 的属性类型如表 2-4 所示。

表 2-4　属性类型

类型	含义
CDATA	纯文本，由可显示字符组成的字符串
（Enumerated）	取值来自一组可接受的取值范围，在( )内被指定
ID	以属性值的方式为文档中的某个元素定义唯一的标识，用以区分具有相同结构相同属性的不同元素
IDREF	属性值引用已定义的 ID 值，方法是把那个元素的 ID 标识值作为该属性的取值
IDREFS	IDREFS 是 IDREF 的复数形式，取值可以是若干个 ID 值，这些 ID 的值用空格隔开
ENTITY	取值为一个已定义的实体
ENTITIES	该属性值包含了多个外部 entity，不同的 entity 之间用空格隔开
NMTOKEN	表示属性值是符合 XML 命名规范的名称
NMTOKENS	表示属性值能够由多个 nmtoken 组成，每个 nmtoken 之间用空格隔开
NOTATION	取值为一个 DTD 中声明的符号，此类型对于使用非 XML 格式的数据非常有用

下面通过实例，对常用的属性类型及属性取值方式进行详细介绍。

1. CDATA

声明为 CDATA 的属性值可以是任何文本串（包括数字和中文），但不能包含“<”，“&”，“"”等字符串，对于下述属性声明：

```
<!ATTLIST student name CDATA #REQUIRED>
```

在 XML 文档中，下列写法都是合法有效的：

```
<student name="Jack"/>
<student name="杰克"/>
<student name="123456"/>
```

下述代码用于实现任务【描述 2.D.3】，创建并使用带属性的内部 DTD，演示 CDATA 的使用。

【描述 2.D.3】 family.xml

```
<?xml version="1.0" encoding="GB2312"?>
<!DOCTYPE family [
    <!ELEMENT family (person+, appliance*)>
    <!ELEMENT person EMPTY>
    <!ELEMENT appliance EMPTY>
    <!ATTLIST person
        name CDATA #REQUIRED
        age CDATA #REQUIRED
        tel CDATA #IMPLIED
>
    <!ATTLIST appliance
        name CDATA #REQUIRED
        quantity CDATA "1"
        comments CDATA #IMPLIED
>
]>
<family>
    <person name="Rose" age="25"/>
    <person name="杰克" age="28" tel="88887777"/>
    <appliance name="冰箱" quantity="2"/>
</family>
```

上述代码中为 person 元素声明了三个属性：name、age 和 tel，其中 tel 可有可无，其他两个属性必须出现；为 appliance 声明了三个属性，comments 可有可无，name 和 quantity 是必须出现的，并且属性 quantity 声明默认值：1。上述 XML 文档合法有效。

如果将 XML 文档的 XML 数据部分写为如下形式：

```
<family>
    <person name="Rose" /> <!-- Error, 缺少 age 属性-->
    <person name="杰克" age="28" tel="88887777"/>
    <appliance name="冰箱" color="red"/><!-- Error, 未定义 color 属性-->
</family>
```

将提示 XML 验证错误信息，原因是此 XML 数据部分声明未遵循其关联的 DTD 文档规范。

2. NMTOKEN/NMTOKENS

某些情况下，可能希望将属性值作为标记，代表某种特殊的意义，如：通过标记信息告知解析器对于此文件的处理级别、安全级别等。此时就需要 XML 中的 NMTOKEN（name token，名称记号），NMTOKEN 是 CDATA 的一个子集，表示属性值必须是英文字母、数字、句点、连字符、下画线或冒号组成，即满足 XML 名称规范。

注意 NMTOKEN 与元素和属性名称并不完全相同，NMTOKEN 的第一个字符可以是任意字符，但建议与 XML 名称规范一致，方便阅读。

下述代码用于实现任务【描述 2.D.4】，创建并使用带 NMTOKEN 属性的内部 DTD，演示 NMTOKEN 的使用。

【描述 2.D.4】 NMTOKENdemo.xml

```
<?xml version="1.0" encoding="GB2312"?>
<!DOCTYPE poems [
    <!ELEMENT poems (poem*)>
    <!ELEMENT poem (title, content)>
    <!ATTLIST poem security_level NMTOKEN #REQUIRED>
    <!ELEMENT title (#PCDATA)>
    <!ELEMENT content (#PCDATA)>
]>
<poems>
    <poem security_level="all">
        <title >黄鹤楼</title>
        <content>
            昔人已乘黄鹤去，此地空余黄鹤楼。黄鹤一去不复返，白云千载空悠悠。
            晴川历历汉阳树，芳草萋萋鹦鹉洲。日暮乡关何处是？烟江波上使人愁。
        </content>
    </poem>
</poems>
```

上述代码表示元素 poem 有一个名为 security_level 的属性，其值符合 XML 名称规则。用户可以通过该属性来控制文档的访问。由于定义属性列表时使用了 NMTOKEN，用户只需创建新值就能适应新的安全级别要求，而不必每次都编辑 DTD。

NMTOKENS 与 NMTOKEN 类似，是 NMTOKEN 的复数，每个 NMTOKEN 之间通过空格隔开。下述代码演示了 NMTOKENS 的使用：

```
<!ATTLIST poem security_level NMTOKENS #REQUIRED>
<poem security_level="role1 role2 role5">
……
</poem>
```

在这里，该 poem 实例可以被 role1、role2 和 role5 访问。就类型而言，这些数据都是有效的 NMTOKEN 值。

注意 NMTOKENS 的值与枚举类型不同，解析器不检查这些 NMTOKENS 取值的有效性，用户必须确保自己使用了适当的名称。

3. ID

ID 类型的属性表明该属性的取值在整个文档中必须是唯一的。它可以作为元素的唯一标识符。每个元素至多有一个 ID 类型的属性。该 ID 必须以一个字母开头。

下述代码用于实现任务【描述 2.D.5】，创建并使用带 ID 属性的内部 DTD，演示 ID 的使用。

【描述 2.D.5】 IDdemo.xml

```
<?xml version="1.0" encoding="GB2312"?>
<!DOCTYPE poems [
    <!ELEMENT poems (poem*)>
    <!ELEMENT poem (content)>
    <!ATTLIST poem
        NO ID #REQUIRED
        title CDATA #REQUIRED>
    <!ELEMENT content (#PCDATA)>
]>
<poems>
    <poem NO="Z001" title="黄鹤楼">
        <content>
            昔人已乘黄鹤去，此地空余黄鹤楼。黄鹤一去不复返，白云千载空悠悠。
            晴川历历汉阳树，芳草萋萋鹦鹉洲。日暮乡关何处是？烟江波上使人愁。
        </content>
    </poem>
    <poem NO="Z002" title="望庐山瀑布">
        <content>
            日照香炉生紫烟，遥看瀑布挂前川。飞流直下三千尺，疑是银河落九天。
        </content>
    </poem>
</poems>
```

上述文档的内部 DTD 将属性 NO 声明为 ID 类型的，XML 的数据部分是合法有效的。如果将两个元素 poem 属性 NO 的取值都改为“Z001”和“Z002”或分别改为“001”和“002”，这两种情况都将提示相关的错误信息。

注意　ID 类型的属性必须设置为#IMPLIED 或#REQUIRED，不能是#FIXED 或默认的，不能为 ID 提供默认值。

4. IDREF/IDREFS

IDREF 属性的值指向文档中其他地方声明的 ID 类型的值。在应用程序中，通过 ID 和 IDREF 实现交叉引用，而不必多次重复整个元素。

IDREF 属性的值与 ID 类型的取值约束一致，而且它必须与文档中的某个 ID 属性具有相

同的值。IDREF 值不能指向文档中不存在的 ID。IDREFS 与 IDREF 类似，是 IDREF 的复数，每个 IDREF 之间通过空格隔开。

下述代码用于实现任务【描述 2.D.6】，创建并使用带 IDREF（IDREFS）属性的内部 DTD，演示 IDREF 的使用。

【描述 2.D.6】 IDREFdemo.xml

```
<?xml version="1.0" encoding="GB2312"?>
<!DOCTYPE family [
    <!ELEMENT family (person+)>
    <!ELEMENT person EMPTY>
    <!ATTLIST person
        relID ID #REQUIRED
        parentID IDREFS #IMPLIED
        name CDATA #REQUIRED
>
]>
<family>
    <person relID="P01" name="Bernie"/>
    <person relID="P02" name="Rose"/>
    <person relID="P03" parentID="P01" name="Joson"/>
    <person relID="P04" name="Tom"/>
    <person relID="P05" name="Barbara"/>
    <person relID="P06" parentID="P04 P05" name="Barbie"/>
</family>
```

上述文档中，通过交叉引用，应用程序（或 XML 解析器）可以通过元素的 parentID 属性找到其引用的 person 元素。

5. 枚举类型

枚举类型（Enumerated）不需要使用关键字，它只是将所有可能的属性取值列举出来，并以竖线（“|”）分隔。枚举类型的每一个可能值都必须遵循 XML 的名称规范。

下述代码用于实现任务【描述 2.D.7】，创建并使用带枚举类型属性的内部 DTD，演示枚举类型的使用。

【描述 2.D.7】 ENUMdemo.xml

```
<?xml version="1.0" encoding="GB2312"?>
<!DOCTYPE students [
    <!ELEMENT students (student+)>
    <!ELEMENT student EMPTY>
    <!ATTLIST student
        name CDATA #REQUIRED
```

```
        age CDATA #REQUIRED
        sex (male | female) "male"
>
]>
<students>
    <student name="Tom" age="17"/>
    <student name="Rose" age="19" sex="female"/>
</students>
```

上述文档中，元素 student 的 sex 属性被声明为枚举类型，可能的取值为 male 或 female，并被设置默认取值为 male。在实际应用中，元素 student 的 sex 属性的取值必须为上述二者取值之一。

2.2.3　实体

实体(Entity)是 XML 中一种可以节省大量时间的机制，它的作用类似于 Word 中的“宏”，也可以将其理解为模板，使用者可以预先定义一个实体，然后在一个文档中多次调用，或者在多个文档中调用同一个实体。简单来说，实体就是 XML 引用一个数据项的方法，它通常是文本，也可以是二进制数据。

一般情况下，实体的命名遵循 XML 名称规范即可，使用实体的好处在于以下两个方面。

- 减少出错率，文档中多个相同的部分只需要输入一遍即可。
- 提高维护效率，比如多个文档都包含地址信息的实体，如果需要修改这个地址信息，只要修改最初定义的实体语句即可。

XML 定义了两种类型的实体：一种是预定义实体；另一种是自定义实体。

1. 预定义实体

在 XML 中，对于 XML 语言定义的某些特殊字符（如尖括号）需保留定义，另外，有些字符是不可打印的。XML 提供了一些预定义的实体，如表 2-5 所示，使用者可以在 XML 文档中利用这些实体表示特定的字符，并保证不产生冲突。因此在元素的文本内容中，可以用实体表示一些特殊字符，以免它们在解析时与文档的标记混淆。

表 2-5　XML 的预定义实体

实体	用途
<	通常用来替换字符小于号（<）
>	通常用来替换字符大于号（>）
&	通常用来替换字符（&）
"	可用来替换字符串中的字符双引号（"）
'	可用来替换字符串中的单引号（'）

在 XML 中，任何字符也都可以表示为数字引用。具体方法是在符号“&#”之后加上字

符的数字值和分号（它们之间没有空格）。例如，大于号也可以表示为“>”。

2. 自定义实体

使用者也可以根据需要在 DTD 中定义一个实体，然后在文档中通过引用来使用它。

自定义实体的语法格式为：

```
<!DOCTYPE rootname [
<!ENTITY entity-name "entity-content">
]>
```

下述代码定义了一段地址信息：

```
<!DOCTYPE ADDRESS [
<!ENTITY ADDRESS "山东路家乐福">
]>
```

如果此地址信息的内容和他人所需的信息来源于同一个文件，也可以使用外部调用的方法。引用格式如下：

```
<!DOCTYPE ADDRESS [
<!ENTITY ADDRESS SYSTEM "http://www.sample.com/ADDRESS.xml">
]>
```

定义好的实体在文档中的引用语法为“&实体名;”。

下述代码用于实现任务【描述 2.D.8】，创建并使用带自定义实体的内部 DTD，演示自定义实体的使用。

【描述 2.D.8】 ENTITYdemo.xml

```
<?xml version="1.0" encoding="GB2312"?>
<!DOCTYPE greeses [
    <!ELEMENT greeses (greese+)>
    <!ELEMENT greese (name, letter, description?)>
    <!ELEMENT name (#PCDATA)>
    <!ELEMENT letter (#PCDATA)>
    <!ELEMENT description (#PCDATA)>
    <!ENTITY alpha "α">
    <!ENTITY beta "β">
]>
<greeses>
    <greese>
        <name>alpha</name>
        <letter>&alpha;</letter>
        <description>第一个希腊字母</description>
    </greese>
```

```
    <greese>
        <name>beta</name>
        <letter>&beta;</letter>
        <description>第二个希腊字母</description>
    </greese>
    <greese>
        <name>小于号</name>
        <letter>&lt;</letter>
    </greese>
</greeses>
```

通过 IE 浏览器查看结果如图 2-1 所示。

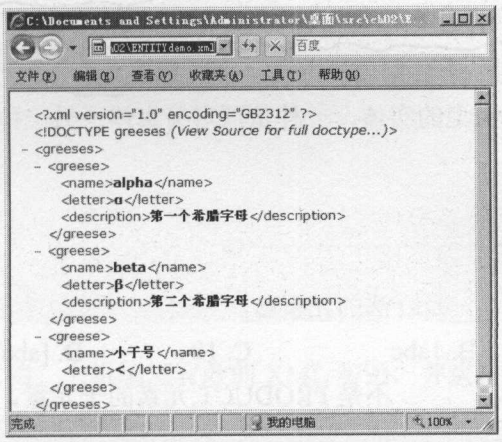


图 2-1　实体显示结果

需要注意的是，实体允许嵌套，如下述代码是允许的：

```
<!ENTITY man "Mr.">
<!ENTITY name "Johnson">
<!ENTITY welcome "&man; &name;">
```

但是，实体不允许循环，如下述代码是错误的：

```
<!ENTITY man "Mr. &name;">
<!ENTITY name "&man; Johnson">
```

当 XML 解析器处理这种情况时，会陷入死循环，用户在定义实体时，务必要严防这种事情的发生。

小结

通过本章的学习，学生应该能够学会：

- DTD 是一套关于标记符的语法规则，是 XML 1.0 规格的一部分。
- DTD 是一种保证 XML 文档格式正确的有效方法。
- DTD 文件是一个 ASCII 的文本文件，后缀名为.dtd。
- DTD 可以在 XML 内部定义也可以在外部定义。
- DTD 提供了 XML 文档所包含的元素、属性和实体及相互关系的定义。
- DTD 的元素类型有 ANY 类型、EMPTY 类型、#PCDATA 类型、父元素类型和混合元素类型。
- 常用的 DTD 属性类型有 CDATA、NMTOKEN/NMTOKENS、ID、IDREF/IDREFS、枚举类型等。
- XML 中实体可以理解为模板，使用者可以预先定义一个实体，然后在一个文档中多次调用，或者在多个文档中调用同一个实体。
- 实体是 XML 引用一个数据项的方法，它通常是文本，也可以是二进制数据。
- XML 定义了两种类型的实体。一种是预定义实体；另一种是自定义实体。



练习

1. 下列选项中，_____是合法的元素名。

   A. TOM　　B. 1abc　　C. 18　　D. [abc]

2. 对于下面的代码，_____不是 PRODUCT 元素的子元素。

```
<!ELEMENT PRODUCT (PRODUCTNAME, DESCRIPTION, PRICE, QUANTITY)>
```

   A. PRODUCTNAME　　B. QUANTITY

   C. DESCRIPTION　　D. NUMBER

3. DTD 文档中，定义属性的关键字是_____。

   A. DOCTYPE　　B. ATTLIST

   C. ELEMENT　　D. ENTITY

4. 下列选项中，_____是预定义实体（多选）。

   A. &　　B. &name;　　C. <　　D. &copyright;

5. DTD 文档中，某元素属性的特点为必须包含该属性，该属性应定义为_____。

   A. #REQUIRED　　B. #IMPLIED

   C. #FIXED value　　D. 默认值

6. 在 DTD 中，元素类型通过_____标记声明，实体类型通过_____标记声明。

7. DOCTYPE 声明由以下部分组成：_____、_____、可选的外部标记符，以及_____。

8. 属性类型设为 ID，表明该属性的取值_____。

9. XML 定义了两种类型的实体，一种是_____；另一种是_____。

10. IDREF 属性的值指向文档中其他地方声明的_____类型的值。

11. 分析下列 XML 实例，上机编写外部 DTD 文档，并进行验证。

```
<?xml version="1.0" encoding="GB2312"?>
<班级>
    <学生 学号="C95001">
        <姓名>李明</姓名>
        <性别>男</性别>
        <年龄>18</年龄>
    </学生>
    <学生 学号="C95002">
        <姓名>张燕</姓名>
        <性别>女</性别>
        <年龄>19</年龄>
    </学生>
    <学生 学号="C95003">
        <姓名>赵彦</姓名>
        <性别>男</性别>
        <年龄>20</年龄>
    </学生>
</班级>
```

12. 分析下列 DTD 文档，上机编写有效的 XML 实例，并进行验证。

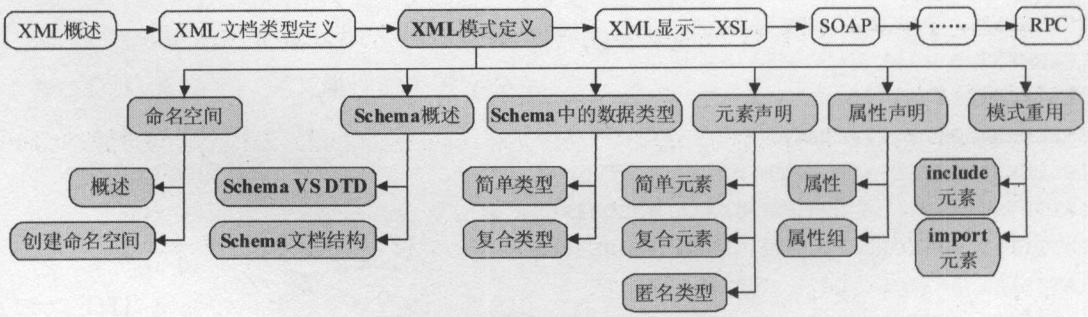
```
<!DOCTYPE NEWSPAPER [
<!ELEMENT NEWSPAPER (ARTICLE+)>
<!ELEMENT ARTICLE (HEADLINE,BYLINE,LEAD,BODY,NOTES)>
<!ELEMENT HEADLINE (#PCDATA)>
<!ELEMENT BYLINE (#PCDATA)>
<!ELEMENT LEAD (#PCDATA)>
<!ELEMENT BODY (#PCDATA)>
<!ELEMENT NOTES (#PCDATA)>
<!ATTLIST ARTICLE AUTHOR CDATA #REQUIRED>
<!ATTLIST ARTICLE EDITOR CDATA #IMPLIED>
<!ATTLIST ARTICLE DATE CDATA #IMPLIED>
<!ATTLIST ARTICLE EDITION CDATA #IMPLIED>
<!ENTITY NEWSPAPER "人民日报">
<!ENTITY PUBLISHER "人民日报出版社">
<!ENTITY COPYRIGHT "人民日报出版社版权所有>
]>
```

第 3 章　XML 模式定义

本章目标

- 了解命名空间的作用
- 掌握命名空间的使用
- 了解 Schema 的作用和特点
- 掌握 Schema 的结构
- 掌握 Schema 的数据类型
- 掌握 Schema 复合数据类型的定义和使用
- 掌握 Schema 的元素定义
- 掌握 Schema 的属性定义
- 掌握 Schema 的模式重用方式

学习导航



任务描述

【描述 3.D.1】

基于命名空间，解决 XML 命名冲突的问题。

【描述 3.D.2】

分别使用 Schema 和 DTD 定义 XML 文档结构。

【描述 3.D.3】

使用 sequence 元素定义 Schema。

【描述 3.D.4】

使用 choice 元素定义 Schema。

【描述 3.D.5】

使用 group 元素定义 Schema。

【描述 3.D.6】

根据给定 XML 文档（student.xml）及相关要求，使用简单元素创建相应 Schema。

【描述 3.D.7】

根据给定 XML 文档（student.xml）及相关要求，使用复合元素创建相应 Schema。

【描述 3.D.8】

对任务【描述 3.D.7】的 Schema 定义进行调整，使用匿名类型定义实现。

【描述 3.D.9】

根据给定的包含带有属性的元素的 XML 文档（student2.xml）结构，创建相应 Schema。

【描述 3.D.10】

根据给定的 XML 文档（student3.xml）结构，使用属性组创建相应 Schema。

【描述 3.D.11】

定义 Schema，使用 include 元素实现模式重用。

【描述 3.D.12】

定义 Schema，使用 import 元素实现模式重用。

3.1 命名空间

在 Java 或 C#等编程语言中，为避免命名冲突、方便维护等原因，提供了包（命名空间）的概念，这对于大规模的程序开发有不可或缺的作用。

在 XML 中，也基于命名空间（namespace）提供了一种避免元素命名冲突的方法。

3.1.1 概述

简单来说，命名空间是一种将程序库名称封装起来的方法，通过这种方法，可以避免应用程序间的命名冲突。

考虑下述两段代码，第一个 XML 文档片段在 table 元素中包含了水果的信息：

```
<table>
    <tr>
        <td>苹果</td>
        <td>葡萄</td>
    </tr>
</table>
```

第二个 XML 文档片段在 table 元素中包含了桌子的信息：

```
<table>
    <name>餐桌</name>
    <width>80</width>
    <length>120</length>
</table>
```

如果需要将上面两个 XML 文档片断合并在一起使用的话，那么将会出现命名冲突的情况。因为这两个片断都包含了<table>元素，而这两个 table 元素的含义是完全不同的。

此时可以引入命名空间以解决上述命名冲突的问题。比如将第一个 XML 文档片段修改为：

```
<fruit:table>
    <fruit:tr>
        <fruit:td>苹果</fruit:td>
        <fruit:td>葡萄</fruit:td>
    </fruit:tr>
</fruit:table>
```

第二个 XML 文档片段修改为：

```
<furniture:table>
    <furniture:name>餐桌</furniture:name>
    <furniture:width>80</furniture:width>
```

```
    <furniture:length>120</furniture:length>
</furniture:table>
```

现在如果把上面的两段代码合为一体的话，就不会再有元素命名冲突的问题了，因为这两个文档对各自的 table 元素使用了不同的前缀，table 元素在两个文档中分别是<fruit:table>和<furniture:table>。所以借助于 XML 命名空间，通过使用前缀的方式，创建了两个不同的 table 元素，从而解决了命名冲突的问题。

3.1.2　创建命名空间

XML 的命名空间本身就是一个 URI（Uniform Resource Identifier，统一资源标识符）。命名空间是阻止具有相同名字元素间冲突的一种方法，在 XML 中，命名空间是被 URI 分配或识别的一个虚拟空间。

命名空间通过给标识名称加一个网址（URL）定位的方法来区别这些名称相同的标识。在 XML 中，命名空间需要在 XML 文档的开头部分声明，一般放置在元素的开始标记（根元素）处，其语法如下所示：

```
xmlns:prefix="URI"
```

其中，prefix 为定义的命名空间的前缀，是可选的；URI 是为当前命名空间选择的网址。

根据是否指定 prefix 可将命名空间分为两种：默认的和明确的。

1. 默认声明

默认命名空间，不需要指定前缀。使用默认声明命名空间的所有元素和属性不需要任何前缀。代码示例如下：

```
<table xmlns="http://www.myweb.com/fruit">
    <tr>
        <td>苹果</td>
        <td>葡萄</td>
    </tr>
</table>
```

2. 明确声明

明确命名空间，xmlns 关键字与一个命名空间 URI 的前缀（prefix）相关联，代码示例如下：

```
<?xml version="1.0" encoding="GB2312"?>
<fruit:table xmlns:fruit="http://www.myweb.com/fruit">
    <fruit:tr>
        <fruit:td>苹果</fruit:td>
        <fruit:td>葡萄</fruit:td>
    </fruit:tr>
```

```
</fruit:table>
```

当在元素的开始标记处使用命名空间时，该元素所有的子元素都将通过一个前缀与同一个命名空间相互关联。

下述代码用于实现任务【描述 3.D.1】，基于命名空间，演示 XML 命名冲突问题的解决。

【描述 3.D.1】 company.xml

```
<?xml version="1.0" encoding="GB2312"?>
<company xmlns="http://www.myweb.com/company"
        xmlns:family="http://www.myweb.com/family">
    <employees>
        <employee>
            <name>张伟</name>
            <age>26</age>
            <phone>66667777</phone>
            <!--开始 family 命名空间-->
            <family:kindred>
                <family:name>张军</family:name>
                <family:phone>22223333</family:phone>
            </family:kindred>
            <!--离开 family 命名空间的范畴-->
        </employee>
    </employees>
</company>
```

上述示例中，定义了两个命名空间，一个默认空间和一个 family 空间，当在定义员工（employee）亲属信息的时候，引入命名空间 family 进行区分，以让 XML 处理器能正确识别。

注意 用来标识命名空间的网络地址并不被 XML 解析器调用，该网络地址的作用仅仅是给命名空间一个唯一的名字，因此这个网络地址也可以是虚拟的，也就是说，设置 URI 并不是指这个标识真的要到那个网址去读取，仅仅作为一种区别的标志而已。使用者可以分配任何名称或字符串作为一个 URI。建议使用一个容易被开发者控制的 URI。

除了可以在根元素定义命名空间外，也可以在任何一个元素中定义，但此时应注意命名空间的范畴问题，必须确保所有来自这个空间的元素都能被定义所在的元素覆盖，如下述代码中，两个子元素 table 元素都使用了 xmlns 属性，使元素和不同的命名空间关联到一起。

```
<?xml version="1.0" encoding="GB2312"?>
<tables>
    <!--fruit 命名空间范畴开始-->
    <fruit:table xmlns:fruit="http://www.myweb.com/fruit">
        <fruit:tr>
```

```
            <fruit:td>苹果</fruit:td>
            <fruit:td>葡萄</fruit:td>
        </fruit:tr>
    </fruit:table>
    <!--离开 fruit 命名空间范畴-->
    <!--furniture 命名空间范畴开始-->
    <furniture:table xmlns:furniture="http://www.myweb.com/furniture">
        <furniture:name>餐桌</furniture:name>
        <furniture:width>80</furniture:width>
        <furniture:length>120</furniture:length>
    </furniture:table>
    <!--离开 furniture 命名空间范畴-->
</tables>
```

注意　建议在根元素中定义命名空间，虽然这会丧失一部分的灵活性，但却大大减少了出错的机会。

3.2　Schema 概述

XML Schema（XML 模式）是 2001 年 5 月正式发布的 W3C 推荐标准，现在已经成为全球公认的 XML 环境下首选的数据建模工具。XML Schema 是 Microsoft 开发的一种定义 XML 文档的模式，称为 XML 模式定义语言——XML Schema Definition，简称 XSD。XML 模式同 DTD 一样是负责定义和描述 XML 文档的结构和内容的模式，它同样可以定义 XML 文档中存在哪些元素和元素之间的关系，并且可以定义元素和属性的数据类型。之所以引入 XML 模式，是因为 DTD 在如下方面存在缺陷。

- DTD 是基于正则表达式的，描述能力有限。
- DTD 没有数据类型的支持，在大多数应用环境下能力不足。
- DTD 的约束定义能力较弱，无法对 XML 实例文档做出更细致的语义限制。
- DTD 不够结构化，重用的代价相对较高。
- DTD 并非使用 XML 作为描述手段，而 DTD 的构建和访问并没有标准的编程接口，无法使用标准的编程方式进行维护。
- DTD 不支持命名空间。

而 XML Schema 正是针对 DTD 的缺点而设计的，XML Schema 的优点如下。

- XML Schema 的语法完全遵循 XML 的语法规范。
- XML Schema 支持一系列的数据类型（整型、浮点型、数据型、布尔型或者许多其他的简单数据类型等）。
- XML Schema 内容模型是开放的，可以随意扩充。

- XML Schema 支持综合命名空间。
- XML Schema 支持属性组。

Schema 相对于 DTD 的最明显好处是 XML Schema 文档本身也是 XML 文档，而不是像 DTD 一样使用自成一体的语法，因此方便用户和开发者使用相同的工具来处理 XML Schema 和其他 XML 信息。另外，Schema 简单易懂，懂得 XML 语法和规则的人都可以立刻理解它。

3.2.1 Schema VS DTD

下述代码用于实现任务【描述 3.D.2】，通过 XML 实例，演示 Schema 与 DTD 的区别。

【描述 3.D.2】note.xml

```
<?xml version="1.0"?>
<!DOCTYPE note SYSTEM "note.dtd">
<note>
    <to>George</to>
    <from>John</from>
    <heading>Reminder</heading>
    <body>Don't forget the meeting!</body>
</note>
```

对于 note.xml 的元素信息，可以定义其 DTD 结构为：

【描述 3.D.2】 note.dtd

```
<!ELEMENT note (to, from, heading, body)>
<!ELEMENT to (#PCDATA)>
<!ELEMENT from (#PCDATA)>
<!ELEMENT heading (#PCDATA)>
<!ELEMENT body (#PCDATA)>
```

如果将此文档结构定义为 XML Schema，其内容为：

【描述 3.D.2】 note.xsd

```
<?xml version="1.0"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
        targetNamespace="http://www.myweb.com"
        xmlns="http://www.myweb.com" elementFormDefault="qualified">
    <xs:element name="note">
        <xs:complexType>
            <xs:sequence>
                <xs:element name="to" type="xs:string"/>
                <xs:element name="from" type="xs:string"/>
```

```
            <xs:element name="heading" type="xs:string"/>
            <xs:element name="body" type="xs:string"/>
        </xs:sequence>
    </xs:complexType>
</xs:element>
</xs:schema>
```

上述定义中，note 元素是一个复合类型，包含<to>、<from>、<heading>、<body>4 个子元素，这些子元素是简易类型，而且必须按顺序出现。

对于 DTD 文档定义，XML 文档的引用方式如 note.xml 的第 2 行所示；XML 对 Schema 的引用将是如下形式：

```
<?xml version="1.0"?>
<note xmlns="http://www.myweb.com"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://www.myweb.com note.xsd">
    <to>George</to>
    <from>John</from>
    <heading>Reminder</heading>
    <body>Don't forget the meeting!</body>
</note>
```

上述代码中，xmlns="http://www.myweb.com"规定了默认命名空间的声明，此声明会告知 Schema 验证器，在此 XML 文档中使用的所有元素都被声明于该命名空间；xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"声明了可用的 XML Schema 实例命名空间；schemaLocation 属性定义了两个值：第一个值是需要使用的命名空间，第二个值是供命名空间使用的 XML Schema 的位置。

3.2.2　Schema 文档结构

通过上述实例看出，XML 模式实际上只是 XML 的另一个应用。一个 XML 模式由导言、不定数量的元素定义和声明组成。

1. 导言

导言在根元素中声明，其一般形式如下：

```
1.  <?xml version="1.0" encoding="GB2312"?>
2.  <xs:schema xmlns:xs=http://www.w3.org/2001/XMLSchema
3.          targetNamespace=http://www.myweb.com
4.          xmlns="http://www.myweb.com"
5.          elementFormDefault="qualified">
6.  <!--Schema 文档中子元素的名字和类型的定义-->
7.  </xs:schema>
```

需要注意，所有 Schema 文档都必须要使用 schema 作为其根元素，如第 2 行代码所示，其中：

- 第 2 行代码指出，这个 XML Schema 中用到的元素和数据类型来自命名空间“http://www.w3.org/2001/XMLSchema”，并且其前缀为 xs。
- 第 3 行代码指出，这个 XML Schema 中定义的元素使用命名空间“http://www.myweb.com”。
- 第 4 行代码指出默认的命名空间是“http://www.myweb.com”。
- 第 5 行代码指出任何 XML 实例文档所使用的且在此 XML Schema 中声明过的元素必须被命名空间限定。

2. 元素定义

一个 XML Schema 由类型定义和元素声明等部件组成，以衡量格式良好的元素信息的有效性，还可以另外规定这些元素项目及其子节点，比如属性和元素的默认值、元素类型、顺序和属性表等。从抽象数据模式上看，XML Schema 的通用组成项是 Schema 组件。一个 XML Schema 是许多 Schema 组件的集合。常用的 Schema 组件有：

- 简单类型定义
- 复合类型定义
- 元素声明
- 属性声明
- 属性组定义
- 标识限定定义
- 模型组定义
- 记法声明

以上组件都有名称，可以单独地出现在 Schema 的内容中，而且可以被单独访问。

3.3 Schema 中的数据类型

XML Schema 中的数据类型可分为简单类型和复合类型，其中简单类型是不能分割的原子类型；复合类型类似于编程语言中的自定义类型，它是由已存在的简单类型组合而成的。

3.3.1 简单类型

XML Schema 规范中定义了两类简单类型：内置类型和用户定义类型。

XML Schema 中的内置数据类型，包括原始（Primitive）数据类型和派生（Derived）数据类型，这些类型是在 XML Schema 中使用的每种数据类型的最基本构成块，可以用它们来

描述元素的内容和属性值，也可以根据这些类型构造自定义的类型。XSD 支持的常用内置数据类型如表 3-1 所示。

表 3-1　内置数据类型

数据类型	描述	示例
string	XML 中任何的合法字符串	God Damn
boolean	逻辑判断，true 或 false	1 和 0 或 true 和 false
number	表示任意精度的十进制数,可使用缩写形式	-1.52，0，1.2E2
float	32 位精确度的浮点实数	123.456
double	64 位精确度的浮点实数	123.456
decimal	任意精度的十进制数	123
long	表示在-2^{63}~$+2^{63}-1$ 之间的一个值	12345
int	表示在-2^{31}~$+2^{31}-1$ 之间的一个值	12345
nonNegativeInteger	表示大于或等于 0 的整数	0，123
positiveInteger	表示一个大于 0 的整数	123，345
dateTime	表示格式为 CCYY-MM_DDThh:mm:ss 的时间	2002-05-14T15:23:45
time	表示 HH:MM:SS 格式的时间	14:12:30
date	表示 CCYY-MM-DD 格式的日期	2001-04-15
hexBinay	表示任意 16 进制编码的二进制数	111100110100
base64Binary	表示任意 base64 编码的二进制数	111110110111
AnyURI	表示一个 URI，可为相对路径或绝对路径	http://www.163.com
Qname	表示一个 XML 命名空间的 Qname	xsd:element
Notation	表示 XML 中的 NOTAITION 类型,不能在模式中直接出现的抽象类型，只能用于派生其他类型	Notation

除了上面介绍的内置数据类型之外，还有一类简单数据类型是用户自定义类型，这种数据类型是编写模式文档的用户对内置类型或其他用户自定义类型加以限制或扩展生成的。用户自定义类型使用<xs:simpleType>元素定义，其语法如下：

```
<xs:simpleType name="自定义数据类型的名称">
    <xs:restriction base="所基于的内置数据类型的名称">
        自定义数据类型的内容模式
    </xs:restriction>
</xs:simpleType>
```

其中，restriction（限定）用于为 XML 元素或者属性定义可接受的值，其可以使用的元素及其含义如表 3-2 所示。

表 3-2　restriction 元素的常用子元素

元素名称	描述
enumeration	在指定的数据集中选择，限定用户的选值
fractionDigits	限定最大的小数位，用于控制精度
length	指定数据的长度
maxExclusive	指定数据的最大值（小于）

（续表）

元素名称	描述
maxInclusive	指定数据的最大值（小于等于）
maxLength	指定长度的最大值
minExclusive	指定最小值（大于）
minInclusive	指定最小值（大于等于）
minLength	指定最小长度
pattern	指定限制数据的正则表达式

例如，记录电话号码的标签格式为：<phoneno>0532-77778888</phoneno>，此标签中的内容要求只能容纳 13 个字符长的字符串值，并且匹配模式 dddd-dddddddd（d 表示 0~9 之间的数字），针对 phoneno 元素的自定义数据类型如下：

```
<xs:simpleType name="phoneno">
    <xs:restriction base="xs:string">
        <xs:length value="13"/>
        <xs:pattern value="\d{4}-\d{8}"/>
    </xs:restriction>
</xs:simpleType>
```

这个自定义的数据类型名为 phoneno；因为只能容纳字符串值，所以必须基于 xs 中的 string 类型进行定义，需要使用 restriction 元素来指出现有的基类型，并且用它来标识约束值的范围，代码如下：

```
<xs:restriction base="xs:string">
    ......
</xs:restriction>
```

在<xs:restriction>元素中间，需要声明此种数据类型的特点，即约束值范围的细节。电话号码要求只能容纳 13 个字符，则代码如下：

```
<xs:length value="13"/>
```

还必须匹配模式 dddd-dddddddd（d 表示 0～9 之间的数字），代码如下：

```
<xs:pattern value="\d{4}-\d{8}"/>
```

其中，\d{4}-\d{8}是一个正则表达式，\d 表示数字，{}用来指定必须显示在表达式中的数字的个数。

上述代码片段中，pattern 元素的属性值必须是一个正则表达式。而正则表达式是由普通字符（例如字符 a 到 z）以及特殊字符（称为元字符）组成的文字模式，它描述了一种字符串匹配的模式，可以用它来检查一个字符串是否含有某种子串、将匹配的子串做替换或者从某个串中取出符合某个条件的子串等。正则表达式可以作为一个模板，将某个字符模式与所

搜索的字符串进行匹配。正则表达式中常用的符号及其含义如表 3-3 所示。

表 3-3　正则表达式常用的符号及含义

字符	描述
\	将下一个字符标记为一个特殊字符或一个原义字符等。例如，'n'匹配字符"n"。'\n'匹配一个换行符。序列'\\'匹配"\"，而"\("则匹配"("
^	匹配输入字符串的开始位置。如果设置了 RegExp 对象的 Multiline 属性，^也匹配'\n'或'\r'之后的位置
$	匹配输入字符串的结束位置。如果设置了 RegExp 对象的 Multiline 属性，$也匹配'\n'或 '\r'之前的位置
*	匹配前面的子表达式零次或多次。例如，zo* 能匹配 "z" 以及 "zoo"。*等价于{0,}
+	匹配前面的子表达式一次或多次。例如，'zo+' 能匹配"zo"以及"zoo"，但不能匹配"z"。+ 等价于 {1,}
?	匹配前面的子表达式零次或一次。例如，"do(es)?"可以匹配"do"或"does"中的"do"。?等价于{0,1}
{n}	n 是一个非负整数。匹配确定的 n 次。例如，'o{2}'不能匹配"Bob"中的'o'，但是能匹配"food"中的两个 o
{n,}	n 是一个非负整数。至少匹配 n 次。例如，'o{2,}'不能匹配"Bob"中的'o'，但能匹配 "foooood"中的所有 o。'o{1,}'等价于'o+'。'o{0,}'则等价于'o*'
{n,m}	m 和 n 均为非负整数，其中 n<=m。最少匹配 n 次且最多匹配 m 次。例如，"o{1,3}"将匹配"fooooood"中的前三个 o。'o{0,1}'等价于'o?'。注意在逗号和两个数之间不能有空格
x|y	匹配 x 或 y。例如，'z|food' 能匹配"z"或"food"。'(z|f)ood'则匹配"zood"或"food"
[xyz]	字符集合。匹配所包含的任意一个字符，例如，'[abc]'可以匹配"plain"中的'a'
[^xyz]	负值字符集合。匹配未包含的任意字符，例如，'[^abc]'可以匹配"plain"中的'p'
[a-z]	字符范围。匹配指定范围内的任意字符，例如，'[a-z]'可以匹配'a'到'z'范围内的任意小写字符
[^a-z]	负值字符范围。匹配任何不在指定范围内的任意字符，例如，'[^a-z]'可以匹配任何不在'a'到'z'范围内的任意字符
\d	匹配一个数字字符，等价于[0-9]
\D	匹配一个非数字字符，等价于[^0-9]

例如，正则表达式"\d{3}-[A-Z]{2}"的语义为：3 个数字后面是一个连字号，接着是两个大写的英文字母。

注意　正则表达式的详细介绍超出了本书的范围，感兴趣的读者可以参阅相关的图书或资料。

用户也可以自定义枚举型的数据类型，语法如下：

```
<xs:simpleType name="自定义数据类型的名称">
    <xs:restriction base="所基于的内置数据类型的名称">
        <xs:enumeration value="枚举值 1"/>
        <xs:enumeration value="枚举值 2"/>
        ......
        <xs:enumeration value="枚举值 n"/>
```

```
    </xs:restriction>
</xs:simpleType>
```

下述代码片段中，定义了一个新类型称为 sex，sex 是从 string 类型导出的，同时它的值只能为 male 或 female，代码如下：

```
<xs:simpleType name="sex">
    <xs:restriction base="xs:string">
        <xs:enumeration value="male"/>
        <xs:enumeration value="female"/>
    </xs:restriction>
</xs:simpleType>
```

下述代码片段则是将名为 num 的数据类型的值设定在 16 到 30 之间：

```
<xs:simpleType name="num">
    <xs:restriction base="xs:positiveInteger">
        <xs:minInclusive value="16"/>
        <xs:maxInclusive value="30"/>
    </xs:restriction>
</xs:simpleType>
```

3.3.2 复合类型

复合类型元素是指包含其他子元素、属性或混合内容的元素。为了声明复合元素，应当首先定义一个复合数据类型（complexType）。然后通过使该数据类型与元素相关联来声明一个复合元素。复合数据类型的声明语法如下所示：

```
<xs:complexType name="数据类型的名称">
    <!--内容模型定义（包括子元素和属性的声明）-->
</xs:complexType>
```

如任务【描述 3.D.2】中 note.xsd 文件中的如下代码片段定义了一个复合数据类型，并将其与元素 note 关联在一起。

```
<xs:element name="note">
    <xs:complexType>
        <xs:sequence>
            <xs:element name="to" type="xs:string"/>
            <xs:element name="from" type="xs:string"/>
            <xs:element name="heading" type="xs:string"/>
            <xs:element name="body" type="xs:string"/>
        </xs:sequence>
    </xs:complexType>
```

```
</xs:element>
```

复合类型经常与元素绑定声明，也可单独声明，并在元素声明中加以引用，此时应为复合类型标注 name 属性。

在 XML 模式中，可以将相关的元素结合为组。Schema 提供了能够用来组合用户定义的元素，常用的元素有以下几个。

- sequence：指定组中的子元素按照一定顺序出现。
- group：用通用名组合成组。
- choice：只能使用组中的子元素之一。
- all：组中的子元素在父元素内可以按照任意顺序出现。

1. sequence 元素

sequence 元素定义一系列元素必须按照模式中指定的顺序显示（如果是可选的，也可以不显示）。其语法格式如下：

```
<xs:sequence
    id=ID
    maxOccurs=nonNegativeInteger|unbounded
    minOccurs=nonNegativeInteger
    any attributes
>
    <!--要组合的子元素的声明-->
</xs:sequence>
```

其中：

- id 规定该元素的唯一的 ID，可选。
- maxOccurs 规定元素在父元素中可出现的最大次数。该值可以是大于或等于零的整数，设置为“unbounded”表示不限制最大次数，默认值为 1。该项可选。
- minOccurs 规定元素在父元素中可出现的最小次数。该值可以是大于或等于零的整数，若设置为 0 则表示该组是可选的，默认值为 1。该项可选。
- any attributes 规定带有 non-schema 命名空间的任何其他属性。该项可选。

下述代码用于实现任务【描述 3.D.3】，定义 Schema，演示 sequence 元素的使用。

【描述 3.D.3】 sequence.xsd

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
    <xs:element name="company">
        <xs:complexType>
            <xs:sequence minOccurs="0" maxOccurs="unbounded">
                <xs:element name="employee" type="xs:string"/>
```

```
            <xs:element name="manager" type="xs:string"/>
        </xs:sequence>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

上述 Schema 定义指明引用该 Schema 的 XML 文档中的<company>元素可以包含零个或多个 employee 元素和 manager 元素，并且要按照先 employee 后 manager 的顺序出现，否则，引用该 Schema 的 XML 文档就不是一个有效的 xml 文档。

需要注意的是，上述 Schema 定义中引入 minOccurs 和 maxOccurs 限定的是复合类型的整体出现次数。也就是如下 XML 文档对上述 Schema 是有效的。

```
<company>
    <employee>Tom</employee>
    <manager>Jonh</manager>
    <employee>White</employee>
    <manager>Rose</manager>
</company>
```

而下列 XML 文档则是无效的。

```
<company>
    <employee>Tom</employee>
    <employee>White</employee>
    <manager>Rose</manager>
</company>
```

如要单独限制每个元素的出现次数，可参考本章 3.4 节介绍的元素声明。

2. choice 元素

choice 元素仅允许包含在<choice>声明中的元素之一出现在包含元素中。其语法格式如下：

```
<xs:choice
    id=ID
    maxOccurs=nonNegativeInteger|unbounded
    minOccurs=nonNegativeInteger
    any attributes
>
    <!--要组合的子元素的声明-->
</xs:choice>
```

其中：

- id 规定该元素的唯一的 ID，可选。

- maxOccurs 规定元素在父元素中可出现的最大次数。该值可以是大于或等于零的整数，设置为“unbounded”表示不限制最大次数，默认值为 1。该项可选。
- minOccurs 规定元素在父元素中可出现的最小次数。该值可以是大于或等于零的整数，若设置为 0 则表示该组是可选的，默认值为 1。该项可选。

下述代码用于实现任务【描述 3.D.4】，定义 Schema，演示 choice 元素的使用。

【描述 3.D.4】 choice.xsd

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
    <xs:element name="company">
        <xs:complexType>
            <xs:choice minOccurs="0" maxOccurs="unbounded">
                <xs:element name="employee" type="xs:string"/>
                <xs:element name="manager" type="xs:string"/>
            </xs:choice>
        </xs:complexType>
    </xs:element>
</xs:schema>
```

上述 Schema 定义指明 company 元素必须至少包含一个 employee 元素或一个 manager 元素，否则，引用该 Schema 的 XML 文档就不是一个有效的 xml 文档。对于该 Schema，如下 XML 文档是有效的。

```
<company>
    <employee>Tom</employee>
    <employee>White</employee>
    <manager>Rose</manager>
</company>
```

3. group 元素

group 元素通过使用通用名将一组元素组合到一起，其语法格式如下：

```
<xs:group
    id=ID
    name=NCName
    ref=QName
    maxOccurs=nonNegativeInteger|unbounded
    minOccurs=nonNegativeInteger
    any attributes
>
    <!--要组合的子元素的声明-->
</xs:group>
```

其中：

- name 用来为组元素分配一个名字，不能包含冒号。该项可选。
- ref 用来在复合元素中引用一个组的名称。该项可选。
- 其他属性与 choice 的对应属性作用一致。

下述代码用于实现任务【描述 3.D.5】，定义 Schema，演示 group 元素的使用。

【描述 3.D.5】 group.xsd

```
<?xml version="1.0"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
    <xs:element name="customer" type="custtype"/>
    <xs:complexType name="custtype">
        <xs:sequence>
            <xs:group ref="custname"/>
            <xs:element name="address" type="addtype"/>
        </xs:sequence>
    </xs:complexType>
    <xs:complexType name="addtype">
        <xs:choice>
            <xs:element name="residence" type="xs:string"/>
            <xs:element name="office" type="xs:string"/>
        </xs:choice>
    </xs:complexType>
    <xs:group name="custname">
        <xs:sequence>
            <xs:element name="firstname" type="xs:string"/>
            <xs:element name="lastname" type="xs:string"/>
        </xs:sequence>
    </xs:group>
</xs:schema>
```

上述 Schema 定义指定 firstname 元素和 lastname 元素组成了一个 custname 元素组，在引用 custname 组时就可以一起引用这两个元素。对于该 Schema，如下 XML 文档是有效的。

```
<customer>
    <firstname>Peter </firstname>
    <lastname>Black</lastname>
    <address>
        <residence>10th  street,BeiJing</residence>
    </address>
</customer>
```

4. all 元素

all 元素常用于 group 元素和 complexType 元素中，它允许所定义的元素可以按照任意顺序显示，all 元素的子元素在默认情况下是必须出现的，而且每次最多显示一次。其语法格式如下：

```
<xs:all
    id=ID
    maxOccurs=1
    minOccurs=0|1
    any attributes
>
    <!--要组合的子元素的声明-->
</xs:all>
```

其中：

- maxOccurs 指定元素可出现的最大次数，该值必须是 1。该项可选。
- minOccurs 指定元素可出现的最小次数，该值可以是整数 0 或 1。若要指定该元素是可选的，请将该属性设置为 0，默认值为 1。该项可选。

注意　all 必须放在任何内容模型的顶层。

如下述 Schema 定义指定 firstname 和 lastname 元素能够以任何顺序出现，两个元素都必须且只能出现一次。

```
<xs:element name="person">
    <xs:complexType>
        <xs:all>
            <xs:element name="firstname" type="xs:string"/>
            <xs:element name="lastname" type="xs:string"/>
        </xs:all>
    </xs:complexType>
</xs:element>
```

而下述 Schema 定义指定 firstname 和 lastname 元素能够以任何顺序出现，每个元素都能出现零次或一次。

```
<xs:element name="person">
    <xs:complexType>
        <xs:all minOccurs="0">
            <xs:element name="firstname" type="xs:string"/>
            <xs:element name="lastname" type="xs:string"/>
        </xs:all>
    </xs:complexType>
```

```
</xs:element>
```

3.4 元素声明

Schema 中的元素声明分为简单元素和复合元素。

3.4.1 简单元素

一个元素中如果仅仅包含数字、字符串或其他数据，不包括子元素和属性，这种元素被称为简单元素，下面的 quantity 元素就是一个简单元素。

```
<quantity>20</quantity>
```

为了在模式文件中声明一个简单元素，必须使该元素与某一简单类型相关联。简单类型可以是 XSD 中内置的数据类型或是用户自定义数据类型。声明简单元素的语法如下：

```
<xs:element name="元素名称"
            type="数据类型"
            default="默认值"
            minOccurs="nonNegativeInteger"
            maxOccurs="nonNegativeInteger|unbounded"/>
```

其中：

- name 指定要声明的元素的名字。
- default 指定该元素默认值，此项可选。
- type 指定该元素的数据类型。
- minOccurs 指定该元素在 XML 文档中可以出现的最小次数，此项可选。
- maxOccurs 指定该元素在 XML 文档中可以出现的最大次数，此项可选。

注意 声明元素时，应该注意以下两点：minOccurs 和 maxOccurs 属性的默认值是 1；minOccurs 如果单独出现，其取值只能为 0 或者 1。maxOccurs 属性如果单独出现，它必须取值为大于等于 1。

下述代码用于实现任务【描述 3.D.6】，根据给定 XML 文档（student.xml）及相关要求，创建相应 Schema。

【描述 3.D.6】 student.xml

```
<?xml version="1.0" encoding="GB2312"?>
<student
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:noNamespaceSchemaLocation="student1.xsd">
```

```
    <name>Tom</name>
    <sex>male</sex>
    <age>19</age>
    <phoneno>88889999</phoneno>
</student>
```

此文档中的元素 name、sex、age 和 phoneno 元素的取值需满足如下要求：

- 元素 name、sex、age 和 phoneno 是简单类型的元素。
- name 元素和 sex 元素的值必须是字符串。
- age 元素的值必须是大于 0 的整数。
- phoneno 元素的值必须是 8 位的数字字符串。

针对上述要求，该 XML 的完整模式文档定义如下：

【描述 3.D.6】 student1.xsd

```
<xs:element name="name" type="xs:string"/>
<xs:element name="sex" type=" xs:string "/>
<xs:element name="age" type=" xs:positiveInteger"/>
<xs:element name="phoneno">
    <xs:simpleType>
        <xs:restriction base="xs:string">
            <xs:length value="8"/>
            <xs:pattern value="\d{8}"/>
        </xs:restriction>
    </xs:simpleType>
</xs:element>
```

上述代码片段中，由于 Schema 的内置数据类型无法满足 phoneno 元素的数据要求，故在声明 phoneno 类型时，通过用户自定义类型加以限制。

3.4.2　复合元素

在 XML Schema 中，包含内容、属性和子元素的元素称之为复合元素。声明复合元素的语法如下：

```
<xs:element name="元素名称"
            type="复合数据类型"
            default="默认值"
            minOccurs="nonNegativeInteger"
            maxOccurs="nonNegativeInteger|unbounded"/>
```

复合元素的声明方式与简单元素的声明方式类似，不同的是 type 属性的值必须通过外部

引用指定一个复合数据类型。

下述代码用于实现任务【描述 3.D.7】，给定 XML 文档（student.xml），根据指定要求，创建相应 Schema。

【描述 3.D.7】 student.xml

```
<?xml version="1.0" encoding="GB2312"?>
<student
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:noNamespaceSchemaLocation="student2.xsd">
    <name>Tom</name>
    <sex>male</sex>
    <age>19</age>
    <phoneno>88889999</phoneno>
</student>
```

此文档中的元素 name、sex、age 和 phoneno 元素的取值需满足如下要求。

- name 元素的值必须是字符串。
- sex 元素的值必须是字符串并且只能取“male”或“female”。
- age 元素的值必须是处于 16 和 30 之间的整数。
- phoneno 元素的值必须是 8 或 11 位的数字字符串。

针对上述要求，该 XML 的完整模式文档定义如下。

【描述 3.D.7】 student2.xsd

```
<?xml version="1.0" encoding="GB2312"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
    <xs:element name="student" type="studentDef"/>
    <!--定义复合类型-->
    <xs:complexType name="studentDef">
        <xs:sequence>
            <xs:element name="name" type="xs:string"/>
            <xs:element name="sex" type="sexDef"/>
            <xs:element name="age" type="ageDef"/>
            <xs:element name="phoneno" type="phonenoDef"/>
        </xs:sequence>
    </xs:complexType>
    <!--定义 age 类型-->
    <xs:simpleType name="ageDef">
        <xs:restriction base="xs:positiveInteger">
            <xs:minInclusive value="16"/>
            <xs:maxInclusive value="30"/>
```

```
        </xs:restriction>
    </xs:simpleType>
    <!--定义 sex 类型-->
    <xs:simpleType name="sexDef">
        <xs:restriction base="xs:string">
            <xs:enumeration value="male"/>
            <xs:enumeration value="female"/>
        </xs:restriction>
    </xs:simpleType>
    <!--定义 phoneno 类型-->
    <xs:simpleType name="phonenoDef">
        <xs:restriction base="xs:string">
            <xs:pattern value="\d{8}|\d{11}"/>
        </xs:restriction>
    </xs:simpleType>
</xs:schema>
```

上述定义中，由于 XML 文档中 student 元素为复合类型，故在 Schema 中为其指定类型时需使用复合类型，studentDef 是按照 XML 文档结构定义的复合数据类型，并被指派（通过 student 元素的 type 属性）为 student 元素的数据类型。

同时，上述文档中，将 age、sex 和 phoneno 的数据类型都扩展为用户自定义数据类型，并分别命名为 ageDef、sexDef 和 phonenoDef，此种形式可以方便地实现数据类型复用。

3.4.3　匿名类型定义

使用 XML Schema，可以通过定义一系列具有名称的数据类型，通过使用元素的 type 属性来引用该数据类型。这种类型的模式构造非常直观，可以方便地实现数据类型复用，但在有些情况下不是很实用。例如，在模式文档中定义了许多只应用一次而且包含非常少的约束的数据类型，那么，使用前面所介绍的方法书写模式文档时就会非常烦琐。对于此种情况，可以使用一种节省名称和外部引用的定义方式——匿名类型定义。

下述代码用于实现任务【描述 3.D.8】，对任务【描述 3.D.7】的 Schema 定义进行调整，使用匿名类型定义实现。

【描述 3.D.8】 student3.xsd

```
<?xml version="1.0" encoding="GB2312"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
    <xs:element name="student">
        <!--匿名定义复合类型-->
        <xs:complexType>
            <xs:sequence>
                <xs:element name="name" type="xs:string"/>
```

```
            <xs:element name="sex">
                <!-匿名定义 sex 类型-->
                <xs:simpleType>
                    <xs:restriction base="xs:string">
                        <xs:enumeration value="male"/>
                        <xs:enumeration value="female"/>
                    </xs:restriction>
                </xs:simpleType>
            </xs:element>
            <xs:element name="age">
                <!--匿名定义 age 类型-->
                <xs:simpleType>
                    <xs:restriction base="xs:positiveInteger">
                        <xs:minInclusive value="16"/>
                        <xs:maxInclusive value="30"/>
                    </xs:restriction>
                </xs:simpleType>
            </xs:element>
            <xs:element name="phoneno">
                <!--匿名定义 phoneno 类型-->
                <xs:simpleType>
                    <xs:restriction base="xs:string">
                        <xs:pattern value="\d{8}|\d{11}"/>
                    </xs:restriction>
                </xs:simpleType>
            </xs:element>
        </xs:sequence>
    </xs:complexType>
</xs:element>
</xs:schema>
```

3.5 属性声明

属性声明用于命名属性并指定属性值的类型。在 Schema 中，实现的方法是使用 attribute 标记，可以按照定义元素的方式定义属性，但受限制的程度较高。它们只能是简单类型，只能包含文本，且没有子属性。属性是没有顺序的，而元素是有顺序的。

3.5.1 属性

在 Schema 中使用 attribute 元素来声明元素的属性，其语法格式如下：

```
<attribute name="属性名"
```

```
        default="默认值"
        fixed="固定值"
        type="数据类型"
        use="optional|required"/>
```

其中：

- name 用来指定自定义属性名称。
- default 用来指定自定义属性的一个默认值。
- fixed 用来为自定义属性提供一个固定的值。
- type 指定该属性的数据类型，此处只能是简单数据类型。
- use 指定该属性值是 required（强制）还是 optional（可选的，默认属性）。

对于下述代码：

```
<xs:attribute name="age" type="xs:integer"/>
```

该语句定义了一个名为 age 的属性，它的值必须是一个整数。把它添加到模式中时，它必须是模式文件中的 schema 元素、complexType 元素或者 attributeGroup 元素的子元素。

注意　要把属性附加在元素上，属性应该在 complexType 定义中的最后进行定义。

下述代码用于实现任务【描述 3.D.9】，根据给定的 XML 文档（student2.xml）结构，创建相应 Schema。

【描述 3.D.9】 student2.xml

```
<?xml version="1.0" encoding="GB2312"?>
<students
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:noNamespaceSchemaLocation="student4.xsd">
    <student name="tom" age="19" sex="male">
        <class>高一二班</class>
    </student>
</students>
```

此文档中的 student 元素有 name、sex 和 age 三个属性，并且需满足如下要求。

- name 元素的值必须是字符串，是必要属性。
- sex 元素的值必须是字符串并且只能取“male”或“female”且可选，默认值为“male”。
- age 元素的值必须是处于 16 和 30 之间的整数且可选。

针对上述要求，该 XML 的完整模式文档定义如下。

【描述 3.D.9】 student4.xsd

```
<?xml version="1.0" encoding="GB2312"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
    elementFormDefault="qualified" attributeFormDefault="unqualified">
    <xs:element name="students">
        <xs:complexType>
            <xs:sequence>
                <xs:element name="student">
                    <xs:complexType>
                        <xs:sequence>
                            <xs:element name="class" type="xs:string"/>
                        </xs:sequence>
                        <!--name 属性，声明该属性为必需属性-->
                        <xs:attribute name="name" type="xs:string"
                        use="required"/>
                        <!--age 属性，使用用户自定义类型进行类型和取值限制-->
                        <xs:attribute name="age">
                            <xs:simpleType>
                                <xs:restriction base="xs:positiveInteger">
                                    <xs:minInclusive value="16"/>
                                    <xs:maxInclusive value="30"/>
                                </xs:restriction>
                            </xs:simpleType>
                        </xs:attribute>
                        <!--sex 属性默认为 male，使用用户自定义类型进行类型和取值限制-->
                        <xs:attribute name="sex" default="male">
                            <xs:simpleType>
                                <xs:restriction base="xs:string">
                                    <xs:enumeration value="male"/>
                                    <xs:enumeration value="female"/>
                                </xs:restriction>
                            </xs:simpleType>
                        </xs:attribute>
                    </xs:complexType>
                </xs:element>
            </xs:sequence>
        </xs:complexType>
    </xs:element>
</xs:schema>
```

由于上述 Schema 定义中，属性 sex 和 age 被声明为 optional（可选的），所以下述两个 XML 结构通过 student4.xsd 验证都是有效的。

```
<student name="tom">
    <class>高一二班</class>
</student>
<student name="John" age="19">
    <class>高一三班</class>
</student>
```

关于属性和元素的默认值，有以下两点需要特别注意。

- 属性和元素的默认值都是使用 default 属性来声明的，但是，在 XML 实例文档中使用时却有所不同。属性和元素的默认值之间的区别是：当属性不出现时，默认的属性值被应用；当元素内容为空的时候，默认的元素值被应用；而元素不出现的时候，默认的元素值不被应用。
- 属性和元素声明中，都使用了 fixed 属性来确保属性和元素被设置为特殊的值。需要注意的是，fixed 值的概念和 default 值的概念是互斥的，如果同时声明 fixed 和 default 属性，那么模式处理器将会报告错误。

3.5.2　属性组

使用 attributeGroup 元素可以把一组属性声明组合在一起，以便定义复合类型时使用，也便于属性在多个元素中重用。其语法格式如下：

```
<attributeGroup name="group name">
    属性 1 的声明
    属性 2 的声明
    ……
    属性 n 的声明
</attrbuteGroup>
```

下述代码用于实现任务【描述 3.D.10】，根据给定的 XML 文档（student3.xml）结构，创建相应 Schema。

【描述 3.D.10】 student3.xml

```
<?xml version="1.0" encoding="GB2312"?>
<students xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:noNamespaceSchemaLocation="student5.xsd">
    <student name="tom" age="19" sex="male">
        <class>高一二班</class>
        <school>实验中学</school>
    </student>
</students>
```

针对上述文档结构，该 XML 的完整模式文档定义如下。

【描述 3.D.10】 student5.xsd

```
<?xml version="1.0" encoding="GB2312"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
    elementFormDefault="qualified" attributeFormDefault="unqualified">
    <xs:element name="students">
        <xs:complexType>
            <xs:sequence>
                <xs:element name="student">
                    <xs:complexType>
                        <xs:group ref="elmtGrp"/>
                        <xs:attributeGroup ref="attGrp"/>
                    </xs:complexType>
                </xs:element>
            </xs:sequence>
        </xs:complexType>
    </xs:element>
    <!--定义元素组-->
    <xs:group name="elmtGrp">
        <xs:sequence>
            <xs:element name="class" type="xs:string"/>
            <xs:element name="school" type="xs:string"/>
        </xs:sequence>
    </xs:group>
    <!--定义属性组-->
    <xs:attributeGroup name="attGrp">
        <xs:attribute name="name" type="xs:string" use="required"/>
        <xs:attribute name="age">
            <xs:simpleType>
                <xs:restriction base="xs:positiveInteger">
                    <xs:minInclusive value="16"/>
                    <xs:maxInclusive value="30"/>
                </xs:restriction>
            </xs:simpleType>
        </xs:attribute>
        <xs:attribute name="sex" default="male">
            <xs:simpleType>
                <xs:restriction base="xs:string">
                    <xs:enumeration value="male"/>
                    <xs:enumeration value="female"/>
                </xs:restriction>
```

```
            </xs:simpleType>
        </xs:attribute>
    </xs:attributeGroup>
</xs:schema>
```

由于上述 Schema 定义中，定义属性组 attGrp 封装了 name、age 和 sex 属性的声明，通过在元素 student 声明中引用该属性组，从而使元素 student 具有上述三个属性。

3.6　模式重用

XML Schema 支持高度重用性，即：在一个模式中声明的组件能够被另一个模式重用。可以通过使用 include 元素和 import 元素来实现模式的重用。

3.6.1　include 元素

include 元素用来包含或引用一个明确地址的外部模式。其语法格式如下：

```
<include id="ID" schemaLocation="filename"/>
```

其中：

- id 用来指定元素的 ID，ID 必须是唯一的。该项可选。
- schemaLocation 指定模式文件的物理地址。

include 元素在一个 xsd 文档中可以多次出现。schema 元素是 include 元素的父元素。在使用 include 元素时，唯一要注意的事情是：要包含和已包含的模式文件都必须属于同一个目标命名空间，如果 schema 目标命名空间不匹配，则包含不会有效。下述代码片段是合法的引用（当该 Schema 文件与 customer.xsd 和 company.xsd 同属一个命名空间时）。

```
<?xml version="1.0"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
    targetNamespace="http://www.myweb.com/schema">
    <xs:include schemaLocation="http://www.myweb.com/schema/customer.xsd"/>
    <xs:include schemaLocation="http://www.myweb.com/schema/company.xsd"/>
    <!-- -->
</xs:schema>
```

下述代码用于实现任务【描述 3.D.11】，定义 Schema，演示如何使用 include 元素实现模式重用。

定义第一个模式文件，内容如下。

【描述 3.D.11】 first.xsd

```
<?xml version="1.0" encoding="UTF-8"?>
<schema xmlns="http://www.w3.org/2001/XMLSchema"
    targetNamespace="www.myweb.com">
    <simpleType name="bid">
        <restriction base="string">
            <pattern value="[A]\d{6}"/>
        </restriction>
    </simpleType>
</schema>
```

上述代码中定义了一种用户自定义的简单类型，并且，对于命名空间使用的是默认方式。目标命名空间：targetNamespace="www.myweb.com"。

定义第二个模式文件，内容如下。

【描述 3.D.11】 second.xsd

```
<?xml version="1.0" encoding="UTF-8"?>
<schema xmlns="http://www.w3.org/2001/XMLSchema"
    targetNamespace="www.myweb.com">
    <simpleType name="aid">
        <restriction base="string">
            <pattern value="[c]\d{6}"/>
        </restriction>
    </simpleType>
</schema>
```

上述代码中也是定义了一种用户自定义的简单类型，并且，对于命名空间使用的是默认的方式。目标命名空间：targetNamespace="www.myweb.com"。

定义第三个模式文件，内容如下。

【描述 3.D.11】 third.xsd

```
<?xml version="1.0" encoding="utf-8"?>
<schema xmlns="http://www.w3.org/2001/XMLSchema"
    xmlns:prd="www.myweb.com" targetNamespace="www.myweb.com">
    <!--指定此模式所在的名称空间，并指明所引用模式所在的名称空间-->
    <include schemaLocation="first.xsd"/>
    <include schemaLocation="second.xsd"/>
    <element name="books" type="prd:infotype"/>
    <complexType name="infotype">
        <sequence>
            <element name="book" type="prd:booktype"/>
```

```
        </sequence>
    </complexType>
    <complexType name="booktype">
        <sequence>
            <element name="title" type="string"/>
            <element name="author" type="prd:atype"/>
        </sequence>
        <attribute name="bookid" type="prd:bid"/>
    </complexType>
    <complexType name="atype">
        <sequence>
            <element name="firstname" type="string"/>
            <element name="lastname" type="string"/>
        </sequence>
        <attribute name="authorid" type="prd:aid"/>
    </complexType>
</schema>
```

在这个模式文件中要用到前两个模式文件中所定义的用户自定义的简单类型；并且，它们必须位于同一个目标命名空间：targetNamespace="www.myweb.com"。对于命名空间 www.myweb.com 中所定义的类型的引用必须加上前缀 prd；而对于 http://www.w3.org/2001/XMLSchema 命名空间中定义的类型的引用不需要加前缀，因为此命名空间使用的是默认引用的方式。

对于上述 Schema 结构，下述 XML 文档是有效的。

【描述 3.D.11】 include.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<prd:books xsi:schemaLocation="www.myweb.com third.xsd"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:prd="www.myweb.com">
    <book bookid="A100001">
        <title>JAVA Programing</title>
        <author authorid="c980002">
            <firstname>White</firstname>
            <lastname>John</lastname>
        </author>
    </book>
</prd:books>
```

3.6.2　import 元素

import 和 include 完成同样功能，但 import 允许访问来自多个不同目标名称空间的模式

的组件。其语法格式如下：

```
<import id="ID" namespace="namespace" schemaLocation="filename"/>
```

具体介绍如下。

- id 用来指定元素的 ID，ID 必须是唯一的，该项可选。
- namespace 指定被引入模式所属的命名空间 URI，它也指定前缀，该前缀用来使一个元素或属性和一个特定的命名空间相关联。
- schemaLocation 指定模式文件的物理地址。

下述代码用于实现任务【描述 3.D.12】，定义 Schema，演示如何使用 import 元素实现模式重用。

定义第一个模式文件，该 Schema 对于命名空间使用的是默认的方式，目标命名空间：targetNamespace="www.first.com"。内容如下所示。

【描述 3.D.12】 first_.xsd

```
<?xml version="1.0" encoding="UTF-8"?>
<schema xmlns="http://www.w3.org/2001/XMLSchema"
    targetNamespace="www.first.com">
    <simpleType name="bid">
        <restriction base="string">
            <pattern value="[A]\d{6}"/>
        </restriction>
    </simpleType>
</schema>
```

定义第二个模式文件，目标命名空间：targetNamespace="www.second.com"。内容如下所示。

【描述 3.D.12】 second_.xsd

```
<?xml version="1.0" encoding="UTF-8"?>
<schema xmlns="http://www.w3.org/2001/XMLSchema"
    targetNamespace="www.second.com">
    <simpleType name="aid">
        <restriction base="string">
            <pattern value="[c]\d{6}"/>
        </restriction>
    </simpleType>
</schema>
```

定义第三个模式文件，内容如下所示。

【描述 3.D.12】 third_.xsd

```
<?xml version="1.0" encoding="utf-8"?>
<schema xmlns="http://www.w3.org/2001/XMLSchema" xmlns:prd="www.myweb.com"
    <!--命名 prd2 指向 www.second.com 命名空间-->
    xmlns:prd2="www.second.com"
    <!--指定此模式所在的命名空间-->
    targetNamespace="www.myweb.com">
    <!--import 中指明所引用模式所在的命名空间-->
    <import namespace="www.second.com" schemaLocation="second_.xsd"/>
    <include schemaLocation="first_.xsd"/>
    <element name="books" type="prd:infotype"/>
    <complexType name="infotype">
        <sequence>
            <element name="book" type="prd:booktype"/>
        </sequence>
    </complexType>
    <complexType name="booktype">
        <sequence>
            <element name="title" type="string"/>
            <element name="author" type="prd:atype"/>
        </sequence>
        <attribute name="bookid" type="prd:bid"/>
    </complexType>
    <complexType name="atype">
        <sequence>
            <element name="firstname" type="string"/>
            <element name="lastname" type="string"/>
        </sequence>
        <!--对于命名空间 www.second.com 中所定义的类型的引用必须加上前缀 prd2-->
        <attribute name="authorid" type="prd2:aid"/>
    </complexType>
</schema>
```

上述模式文件用到前两个模式文件中所定义的用户自定义简单类型；因为它们并不位于同一个目标命名空间，所以需要借助于 import 元素将其他目标命名空间中的模式文件引入当前的模式文件中。

third_.xsd 模式文件所在的目标命名空间与 first_.xsd 模式文件是一样的，因此，引入 first_.xsd 模式文件时使用的是 include 元素；而 third_.xsd 模式文件所在的目标命名空间与 second_.xsd 模式文件不是一样的，因此，引入 second_.xsd 模式文件时使用的是 import 元素。

对于命名空间 www.myweb.com 中所定义的类型的引用必须加上前缀 prd；对于命名空间 www.second.com 中所定义的类型的引用必须加上前缀 prd2；而对于 http://www.w3.org/2001/

XMLSchema 命名空间中定义的类型的引用就不需要加前缀了，因为此名称空间使用的是默认引用的方式。

对于上述 Schema 结构，下述 XML 文档是有效的。

【描述 3.D.12】 import.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<prd:books xsi:schemaLocation="www.myweb.com third_.xsd"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:prd="www.myweb.com">
    <book bookid="A100001">
        <title>JAVA Programing</title>
        <author authorid="c980002">
            <firstname>White</firstname>
            <lastname>John</lastname>
        </author>
    </book>
</prd:books>
```

小结

通过本章的学习，学生应该能够学会：

- 命名空间是防止 XML 命名冲突的有效方式。
- 在 XML 中，命名空间是被统一资源标识符 URI 分配或识别的一个虚拟空间。
- 用来标识命名空间的网络地址无须具有实际意义。
- Schema 是替代 DTD 的模式定义语言。
- XML Schema 的语法完全遵循 XML 的语法规范。
- Schema 由导言、不定数量的元素定义和声明组成。
- 所有 Schema 文档都必须要使用 schema 作为其根元素。
- XML Schema 中的数据类型可分为简单类型和复合类型。
- 复合类型元素通过 complexType 声明，包含其他元素、属性和混合内容。
- 常用于定义复合类型的元素有 sequence、choice、group、all。
- XML Schema 中的数据类型可分为简单类型和复合类型。
- 复合元素的 type 属性值必须通过外部引用指定一个复合数据类型。
- Schema 声明的属性是没有顺序的，只能是简单类型，只能包含文本，且没有子属性。
- 使用 attributeGroup 元素可以把一组属性声明组合在一起，以便定义复合类型时使用。
- include 元素用于包含同一个命名空间下的 Schema 文件。
- import 元素用于包含不同命名空间下的 Schema 文件。

练习

1. XML_____提供了一种避免元素命名冲突的方法。

   A. 命名空间　　B. DTD

   C. XSD　　D. XSL

2. 下述关于命名空间的描述，正确的是_____。（多选）

   A. 只能在根元素定义命名空间

   B. 可以在根元素以外的其他元素定义命名空间

   C. 用于定义命名空间的 URI 必须有实际意义

   D. 默认的命名空间不需要指定前缀

3. 下述关于 Schema 的描述，错误的是_____。

   A. XML Schema 的语法完全遵循 XML 的语法规范

   B. XML Schema 内容模型是开放的，可以随意扩充

   C. XML Schema 不支持综合命名空间

   D. XML Schema 支持属性组

4. 下述关于 DTD 的描述，错误的是_____。

   A. DTD 是基于正则表达式的，描述能力有限

   B. DTD 没有数据类型的支持，在大多数应用环境下能力不足

   C. DTD 的约束定义能力不足，无法对 XML 实例文档作出更细致的语义限制

   D. DTD 支持命名空间

5. 定义一个元素组，要求一次只能出现组中的一个元素，要使用的元素是_____，规定一列元素需按顺序出现的元素是_____。

   A. sequence　　B. group　　C. choice　　D. all

6. 通过使用_____关键字，在 XSD 中声明名称空间，根据是否指定前缀可以将命名空间分为_____和_____。

7. 所有 Schema 文档都必须要使用_____作为其根元素。

8. 声明用户自定义的简单数据类型使用关键字_____，声明复合数据类型使用关键字_____。

9. XSD 中的元素_____允许引用任何在同一目标命名空间的上下文中被定义的外部模式，_____允许引用不同命名空间的模式文件。

10. 对于下列 XML 文档：

```
<?xml version="1.0" encoding="GB2312"?>
<Orders>
    <Order orderID="A001" orderDate="2010-7-20">
        <name>玩具</name>
        <number>16</number>
```

```
        <city>上海</city>
        <zip>200000</zip>
        <phoneno>13577778888</phoneno>
    </Order>
    <Order orderID="A002" orderDate="2010-7-22">
        <name>文具</name>
        <number>17</number>
        <city>青岛</city>
        <zip>266000</zip>
        <phoneno>0532-66667777</phoneno>
    </Order>
</Orders>
```

创建一个 Schema，并应用于给定的 XML 文档。要求如下：

- Order 元素在 XML 文档中可以出现多次，但是至少要出现一次。
- OrderID 值的格式必须是 AXXX，其中 X 为 0～9 的数字。
- number 的值要在 1～999 之间。
- zip 元素的内容的格式必须是 XXXXXX，其中 X 为 0～9 的数字，该元素可选。
- phoneno 允许如下组合：11 位手机号、3 位区号+8 位号码、4 位区号+8 位号码。

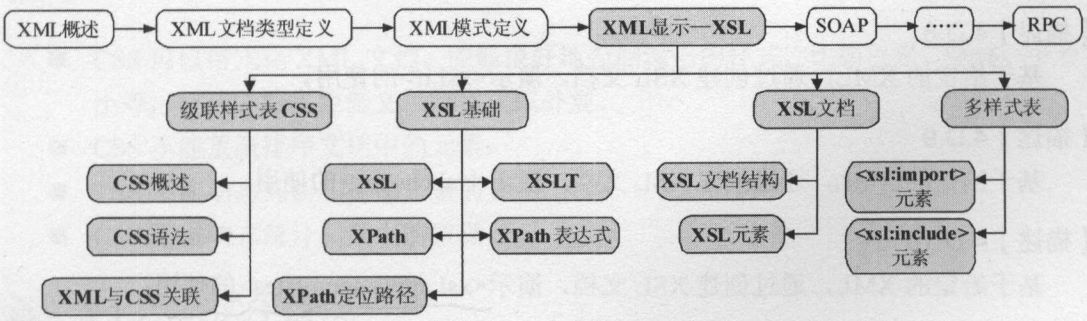
11. 简要描述 DTD 的特点，并说明使用 Schema 代替 DTD 的原因。

第 4 章　XML 显示——XSL

本章目标

- 掌握 XML 与 CSS 的关联
- 了解 XSL 的基本结构
- 了解 XSL 处理 XML 文档的过程
- 了解 XPath 的特点和结构
- 掌握 XPath 的数据类型和结点间的关系
- 掌握 XPath 表达式及应用
- 掌握 XPath 定位路径的使用
- 掌握常用 XSL 元素的使用
- 掌握 XSL 多样式表的处理方式

学习导航



任务描述

【描述】4.D.1

定义 CSS 样式并编写相应的 XML 文档，演示 CSS 对 XML 显示的控制。

【描述】4.D.2

使用微软的 XML DOM 对象来载入 XML 文档，并使用 selectNodes()函数从 XML 文档选取结点，演示 XPath 的使用。

【描述】4.D.3

基于给定的 XML，通过创建 XSL 文档，理解 XSL 的基本结构和显示效果。

【描述】4.D.4

基于给定的 XML，通过创建 XSL 文档，演示<xsl:template>的使用。

【描述】4.D.5

基于给定的 XML，通过创建 XSL 文档，演示<xsl:value-of>的使用。

【描述】4.D.6

基于给定的 XML，通过创建 XSL 文档，演示<xsl:for-each>的使用。

【描述】4.D.7

基于给定的 XML，通过创建 XSL 文档，演示<xsl:sort>的使用。

【描述】4.D.8

基于给定的 XML，通过创建 XSL 文档，演示<xsl:if>的使用。

【描述】4.D.9

基于给定的 XML，通过创建 XSL 文档，演示<xsl:choose>的使用。

【描述】4.D.10

基于给定的 XML，通过创建 XSL 文档，演示<xsl:apply-templates>的使用。

【描述】4.D.11

创建 XSL 文档，演示<xsl:import>元素的使用。

【描述】4.D.12

创建 XSL 文档，演示<xsl:include>元素的使用。

4.1　级联样式表 CSS

在 XML 文件中，使用的是自定义的标记，浏览器仅仅是作为一个 XML 文件的解析器，无法理解这些标记的含义，只要 XML 文件是 Well-Formed 的，那么它就将文件原封不动地显示出来。由于编辑 XML 文档时，仅仅需要关注文件的内容、信息的结构，XML 数据本身并不包含数据显示的信息，如果希望让 XML 文档像 Web 页面那样显示，就必须添加一些额外的显示信息。这些包含控制 XML 文档显示信息的文档就是样式表，样式表实现了数据或内容与表示层的分开。

W3C 推荐的样式表标准有两个：CSS 级联样式表和 XSL 可扩展样式表。

4.1.1　CSS 概述

级联样式表 CSS（Cascading Style Sheets）是一种文本文档，其包含特殊元素格式特征的规则和定义，可用于控制 XML 文档和 HTML 文档中的标签如何被格式化。即：一个 CSS 样式表就是一组规则，每个规则给出此规则所适用的元素的名称，以及此规则要应用于目标元素的样式控制。

目前，CSS 现在已经被大多数的浏览器所支持，成为网页设计必不可少的工具之一。使用 CSS 能够简化网页的格式代码，加快下载显示的速度，也减少了需要上传的代码数量，大大减少了重复劳动的工作量。

使用 CSS 样式表可以方便地给 XML 文档添加额外的显示信息，实现数据与数据的表现形式的分离。CSS 与 XML 结合比与 HTML 结合得更好。由于 XML 元素没有任何预定义的格式规定，所以不会限制何种 CSS 样式只能用于何种元素。但是，CSS 与 XML 结合时也有局限性。

- CSS 可以格式化 XML 文档，能够很好地控制输出的样式，比如色彩、字体、字号大小等，但只能静态控制文档的样式或外观。
- CSS 不能重新排序文档中的元素。
- CSS 不能动态判断和控制元素的显示。
- CSS 无法动态统计元素中的数据。

4.1.2　CSS 语法

CSS 的定义由三个部分构成：选择符（selector）、属性（properties）和属性的取值（value）。其语法格式如下：

```
selector
{
property1: value;
property2: value;
```

```
.....
propertyN: value;
}
```

具体介绍如下。

- selector 是选择符，最普通的选择符就是元素的名称。可以用逗号将选择符中的所有元素分开，把一组属性应用于多个元素，这样可以减少样式重复定义，例如：

```
h1,h2,h3,h4,h5,h6 { color: green }
p,table{ font-size: 9pt }
```

- property1、property2 和 propertyN 为属性名。
- value 是为对应属性名指定的值。
- 每对属性名/属性值后一般要跟一个分号（只有一对名/值的情况除外）。

如下述 CSS 代码片段：

```
BOOKNAME
{
font-family:Arial;
font-size:15pt;
font-weight:bold;
color:red;
display:block;
}
```

此样式表中定义了一个规则，这个规则指定 BOOKNAME 元素应以 15 磅（font-size:15pt）、粗体（font-weight:bold）的 Arial 字体（font-family:Arial）并将其内容以红颜色（color:red）显示在块中(display:block)。

4.1.3 XML 与 CSS 关联

使用 CSS 样式表可以方便地给 XML 文档添加额外的显示信息。在 XML 中通过 xml-stylesheet 指令引入 CSS 样式表，其语法格式如下：

```
<?xml-stylesheet type="text/css" href="location" ?>
```

其中：

- <?xml-stylesheet?>处理指令中的 type 属性是正在使用的样式表的类型。对 CSS 来说，其值是 text/css，对 XSL 来说，则是 text/xsl。
- <?xml-stylesheet?>处理指令中的 href 属性值是一个 URL，通常是相对值，指明在何处可找到样式表。如果样式表没有找到，Web 浏览器很可能使用其默认的样式表，

此时浏览器可能会报告一条错误信息。

下述代码用于实现任务【描述 4.D.1】，定义 CSS 样式并编写相应的 XML 文档，演示 CSS 对 XML 显示的控制。

给定 CSS 样式内容如下。

【描述 4.D.1】 product.css

```
PRODUCTNAME
{font-family:Arial;
font-size:15pt;
font-weight:bold;
color:red;
display:block;
padding-top:6pt;
padding-bottom:6pt
}
PRICE,DESCRIPTION,QUANTITY
{font-family:Arial;
font-size:8pt;
color:teal;
display:block;
padding-top:3pt;
padding-bottom:3pt
}
```

上述 CSS 中，将 PRICE、DESCRIPTION 和 QUANTITY 元素分为一组，统一定义其样式。对于上述 CSS，编写相应的 XML 文档如下。

【描述 4.D.1】 product.xml

```
<?xml version="1.0" encoding="gb2312"?>
<?xml-stylesheet type="text/css" href="product.css"?>
<PRODUCTS>
    <PRODUCT PRODID="p001" CATEGORY="Toy">
        <PRODUCTNAME>模型飞机</PRODUCTNAME>
        <DESCRIPTION>适合于 5 岁以上儿童</DESCRIPTION>
        <PRICE>239</PRICE>
        <QUANTITY>20</QUANTITY>
    </PRODUCT>
    <PRODUCT PRODID="p002" CATEGORY="Toy">
        <PRODUCTNAME>电动汽车</PRODUCTNAME>
        <DESCRIPTION>适合 4 岁以上儿童</DESCRIPTION>
        <PRICE>860</PRICE>
```

```
        <QUANTITY>12</QUANTITY>
    </PRODUCT>
</PRODUCTS>
```

上述代码中第二行是<?xml-stylesheet?>处理指令，此指令通知 Web 浏览器加载本文档，以便在文件 product.css 中找到的样式表应用于本文档。

通过 IE 浏览器查看 product.xml，结果如图 4-1 所示。

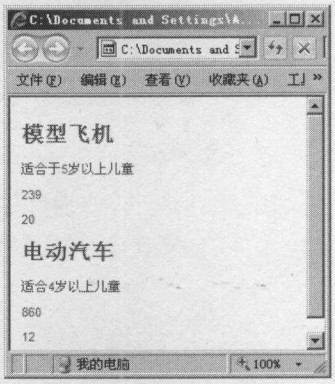


图 4-1 关联 CSS 运行结果

注意 关于 CSS 详细的技术内容超出本书范围。如读者想详细学习 CSS，可参考相关 CSS 的图书或资料。

4.2 XSL 基础

CSS 只能处理简单的、顺序固定的 XML 文件，对于复杂的、高度结构化的 XML 文档的处理就无能为力了，W3C 组织推出了 XSL 用于解决此类难题。

4.2.1 XSL

XSL（eXtensible Stylesheet Language，可扩展样式语言）是一种基于 XML 的语言，用于显示 XML 文件的一种规范。与 CSS 不同，XSL 是遵循 XML 的规范来制定的，即 XSL 文件本身符合 XML 的语法规定。如果说 CSS 是 HTML 的专用样式表，XSL 就是 XML 的专用样式表。同 CSS 相比，XSL 在排版样式的功能上要比 CSS 强大。

XSL 具有以下功能。

- 能够将 XML 转换成 HTML 进行显示。
- 可以过滤和分类 XML 中的数据。
- 可以对 XML 文档中的元素进行寻址，并决定哪些元素被显示，显示几次。
- 可以基于数据值格式化 XML 数据（如用红色显示负数）。

XSL 由两部分组成：一是转化 XML 文档（XSLT）；二是格式化 XML 文档（XSLFO）。XSL 组成如图 4-2 所示。

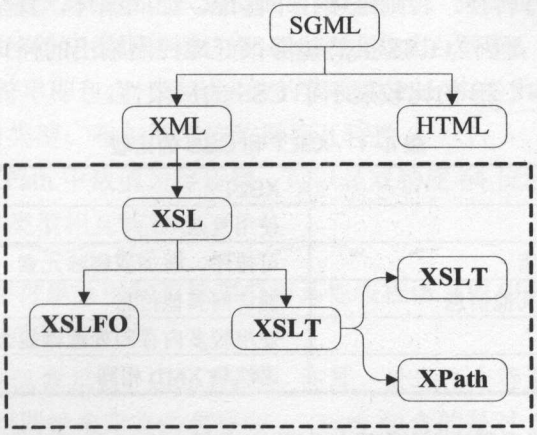


图 4-2　XSL 组成

XSL 实际上包含三种语言，XSLT、XPath 和 XSLFO，具体介绍如下。

- XSLT 是一种转换 XML 的语言，着重于 XML 文档转换。
- XPath 是一种用来访问 XML 文档不同部分或模式的语言，着重于从 XML 层次结构上访问结点。
- XSLFO（XSL 格式化对象）是一种定义 XML 显示方式的语言，着重于格式对象，作用就类似 CSS 在 HTML 中的作用。

注意　在 XSL 中最重要的是 XSLT，而 XSLFO 不在本书介绍范围之内。

4.2.2　XSLT

XSLT（eXtensible Stylesheet Language Transformation）是一种用来转换 XML 文档结构的语言。XSL 和 XSLT 从狭义上理解是一样的，而按照 W3C 的标准，XSLT 的说法更严格一些。

XSL 的处理过程是这样的：XML 文件在展开后是一种树状结构，称为“原始树”，XSL 处理器从这个树状结构读取信息，根据 XSL 样式的指示对这个“原始树”进行排序、复制、过滤、删除、选择、运算等操作后产生另外一个“结果树”（DOM 树），然后在“结果树”中加入一些新的显示控制信息，如表格、其他文字、图形，以及一些有关显示格式的信息。XSL 处理器根据 XSL 样式表的指示读取 XML 文件中的信息，然后再重新组合后转换产生一个格式良好的 HTML 文件。

总的来说，使用 XSL 样式表对 XML 文档进行转换的过程分为以下两步。

- 首先是根据 XML 文档构造源树，然后根据 XSL 规则将源树转换为结果树。
- 生成结果树后，对其进行解释，格式化为一种适合显示、打印或是播放的格式。

XSL 处理器负责实现转换过程。首先，XML 文档被解析成 DOM 树存放在内存中，接着对文档进行分析，每一个 DOM 树中的结点都会与一个模板相比较，当二者匹配时，就会按照模板中定义的规则进行转换，否则继续往下匹配。如此循环，直至整个文档处理完毕。

之所以引入 XSLT，是因为 CSS 虽然能够很好地控制输出的样式，但是它有严重的局限性，表 4-1 通过 XSLT 和 CSS 的比较展示了 CSS 的局限性。

表 4–1　XSLT 和 CSS 的比较

CSS	XSLT
使用简单	使用复杂
不能排序、添加或删除元素	可排序、添加或删除元素
不能访问文档除元素外的其他信息	能访问其他信息
使用内存比较少	使用较多内存和处理器能力
与 XML 语法不同	语法与 XML 相同

总之，CSS 只适合用于输出比较固定的最终文档。CSS 的优点是简洁，消耗系统资源少；而 XSLT 功能强大，但因为要重新索引 XML 结构树，所以占用内存较多；而在 XML 中创建 Web 页面的有用技术，是用 XSLT 将 XML 转换为诸如列表或表的结构，然后将 CSS 应用于结果，控制这些结构在适当媒体中的显示。因此，常常将它们结合起来使用，如在服务器端用 XSLT 处理文档，在客户端用 CSS 来控制显示，可以减少响应时间。

4.2.3　XPath

XML 是一个完整的树状结构文档，在转换 XML 文档时需要按照某种规则处理 XML 的数据（结点）。XPath 就是一种专门用来在 XML 文档中定位和查找信息的语言，通过 XPath 可对 XML 文档中的元素和属性进行遍历，是很多高级 XML 应用的基础。XPath 隶属 XSLT，是 W3C XSLT 标准的主要元素，因此通常会将 XSLT 语法和 XPath 语法混在一起介绍。

可以这样理解 XPath，如果将 XML 文档看做是一个数据库，XPath 就是 SQL 查询语言；如果将 XML 文档看做是 DOS 目录结构，XPath 就是 cd、dir 等目录操作命令的集合。

XPath 查询语言具有如下特点。

- XPath 使用路径识别 XML 元素，这些路径表达式看起来很像计算机的文件系统。
- XPath 定义了一个标准函数库，可以帮助精确寻找需要的结点。
- XPath 是 XSLT 的一个主要元素；在 XSLT 文档中，XPath 主要用于 match、select、test 属性。
- XPath 是一个 W3C 标准，它也可以被用于 XPointer 等其他 XML 解析软件。
- XPath 并不使用 XML 规则来构造表达式，它有着自己的规则。

1. XPath 数据类型

XPath 主要有以下四种数据类型。

- 结点集：结点集是通过路径匹配返回符合条件的一组结点的集合，其他类型的数据不能转换为结点集。
- 布尔类型：由函数或布尔表达式返回的条件匹配值，有 true 和 false 两个值。布尔类型和数值类型、字符串类型可通过 boolean()函数相互转换。
- 字符串类型：字符串即包含一系列字符的集合，XPath 中提供了一系列的字符串函数，字符串可与数值类型、布尔类型的数据相互转换。
- 数值类型：在 XPath 中数值为浮点数，可以是双精度 64 位浮点数，数值也可以和布尔类型、字符串类型相互转换。

其中，布尔类型、字符串类型和数值类型与其他编程语言中相应的数据类型差不多，但是结点集是 XML 文档树的特有产物。

一个 XML 文件可以包含元素、CDATA 节、注释、处理指令等逻辑要素，其中元素还可以包含属性，并可以利用属性来定义命名空间，XPath 包含的是对 XML 文档结构树的一系列操作，因此，相应地在 XPath 中，可以将结点划分为以下七种结点类型。

（1）根结点（Root Node）

根结点是一棵树的最上层，根结点是唯一的。树上其他所有元素结点都是它的子结点或后代结点。根结点与 xml 文档中的根元素是不同的概念，根元素是根结点的第一层子结点。根结点一般包括两个子结点：xml-stylesheet 处理指令和根元素。对根结点的处理机制与其他结点相同。在 XSLT 中对树的匹配总是先从根结点开始。

（2）元素结点（Element Nodes）

元素结点对应于文档中的每一个元素，一个元素结点的子结点可以是元素结点、注释结点、处理指令结点和文本结点，可以为元素结点定义一个唯一的标识 id。元素结点都可以有扩展名，它由两部分组成：一部分是命名空间 URI，另一部分是本地的命名。

（3）文本结点（Text Nodes）

文本结点包含了一组字符数据，即 CDATA 中包含的字符。任何一个文本结点都不会有紧邻的兄弟文本结点，而且文本结点没有扩展名。

（4）属性结点（Attribute Nodes）

每一个元素结点有一个相关联的属性结点集合，元素是每个属性结点的父结点，但属性结点却不是其父元素的子结点。这就是说，通过查找元素的子结点可以匹配出元素的属性结点，但反过来不成立，只是单向的。另外，元素的属性结点没有共享性，也就是说不同的元素结点不共有同一个属性结点。

注意　对默认属性的处理等同于定义了的属性。如果一个属性是在 DTD 声明的，但声明为#IMPLIED，而该属性没有在元素中定义，则该元素的属性结点集中不包含该属性。此外，与属性相对应的属性结点都没有命名空间的声明。命名空间属性对应着另一种类型的结点。

（5）命名空间结点（Namespace Nodes）

每一个元素结点都有一个相关的命名空间结点集。在 XML 文档中，命名空间是通过保留属性声明的，因此，在 XPath 中，该类结点与属性结点极为相似，它们与父元素之间的关系是单向的，并且不具有共享性。

（6）处理指令结点（Processing Instruction Nodes）

处理指令结点对应于 XML 文档中的每一条处理指令。它也有扩展名，扩展名的本地命名指向处理对象，而命名空间部分为空。

（7）注释结点（Comment Nodes）

注释结点对应于文档中的注释。

2. XPath 结点关系

XPath 中，结点之间的关系有父（parent）、子（children）、同胞（sibling）、先辈（ancestor）和后代（descendant）。

如对于下述 XML 文档：

```
<?xml version="1.0" encoding="GB2312"?>
<bookstore>
    <book>
        <title lang="en">RESTful Web Services</title>
        <author>Leonard Richardson</author>
        <year>2007</year>
        <price>29.00</price>
    </book>
</bookstore>
```

- book 元素是 title、author、year 和 price 元素的父结点。每个元素以及属性都有一个父结点。
- title、author、year 和 price 元素都是 book 元素的子结点。元素结点可有零个、一个或多个子结点。
- title、author、year 和 price 元素都是同胞结点。同胞结点拥有相同的父结点。
- title 元素的先辈是 book 元素和 bookstore 元素。先辈可以是某结点的父结点、父结点的父结点等。
- bookstore 的后代是 book、title、author、year 和 price 元素。后代可以是某个结点的子结点，子结点的子结点等。

4.2.4 XPath 表达式

XPath 将 XML 文档看做由结点构成的层次树。每棵树包括：元素结点、属性结点、文本结点、处理指令结点、注释结点和命名空间结点。可以通过编写 XPath 表达式来定位树中特定的结点，XPath 使用路径表达式在 XML 文档中选取结点，表 4-2 描述了常用的路径表达式。

表 4-2　XPath 路径表达式

表达式	描述
nodename	选取此结点的所有子结点
/	从根结点选取
//	从匹配选择的当前结点选择文档中的结点，而不考虑它们的位置
.	选取当前结点
..	选取当前结点的父结点
@	选取属性

其中，“/”和“//”是 XPath 表达式用来定位结点的两种表示方法，“/”代表这是绝对路径，表示当前文档的根结点，类似 DOS 目录分割符；“//”则表示相对路径，表示当前文档所有的结点，类似查看整个目录，此时文件中所有符合模式的元素都会被选出来，即使是处于树中不同的层级也会被选出来。

下述示例讲解将基于如下 XML 文档。

```
<?xml version="1.0" encoding="GB2312"?>
<bookstore>
    <book>
        <title lang="en">RESTful Web Services</title>
        <price>29.00</price>
    </book>
    <book>
        <title lang="zh">Java 编程基础</title>
        <price>46.00</price>
    </book>
</bookstore>
```

表 4-3 列举了部分 XPath 表达式实例。

表 4-3　XPath 表达式实例

路径表达式	描述
bookstore	选取 bookstore 元素的所有子结点
/bookstore	选取根结点 bookstore 元素
/bookstore/book/price	选取 bookstore 元素下所有 book 元素的所有 price 元素
/bookstore/book/*	选取/bookstore/book 的所有子元素
bookstore/book	选取 bookstore 元素下所有的 book 子元素
//bookstore	选取文档中所有的 bookstore 元素，无论它在什么层次
bookstore//book	选取在 bookstore 元素下所有的 book 元素，无论它们位于 bookstore 之下的什么位置
/bookstore/*/price	选取 bookstore 的子元素中，包含有 price 作为子元素的元素
//*	选取文件中的所有元素
//@lang	选取所有名为 lang 的属性

注意 想要存取不分层级的元素，XPath 语法必须以两个斜线开头(//)，想要存取未知元素才用星号(*)，星号只能代表未知名称的元素，不能代表未知层级的元素。

XML 中，可以使用谓语来查找某个特定的结点或者包含某个指定的值的结点，谓语被嵌在方括号中。下面的 XPath 表达式表示从 bookstore 的子元素中取出第一个叫做 book 的元素。XPath 中第 1 个元素的下标从 1 开始。

```
/bookstore/book[1]
```

表 4-4 列出了带有谓语的一些路径表达式，以及表达式的结果。

表 4-4 XPath 谓语表达式实例

路径表达式	描述
/bookstore/book[last()]	选取属于 bookstore 子元素的最后一个 book 元素
/bookstore/book[last()-1]	选取属于 bookstore 子元素的倒数第二个 book 元素
/bookstore/book[position()<3]	选取前两个属于 bookstore 元素的子元素的 book 元素
//title[@lang]	选取所有拥有名为 lang 的属性的 title 元素
//title[@lang="en"]	选取所有 title 元素，且这些元素拥有值为 eng 的 lang 属性
/bookstore/book[price>15.00]	选取所有 bookstore 元素的 book 元素，且其中的 price 元素的值须大于 15.00
/bookstore/book[price>15.00]/title	选取所有 bookstore 元素中的 book 元素的 title 元素，且其中的 price 元素的值须大于 15.00
//title[@*]	选取所有带有属性的 title 元素

如果需要选择一个以上的路径，可以在 XPath 表达式中使用“|”运算符。表 4-5 描述了一些路径表达式，以及这些表达式的结果。

表 4-5 XPath 路径表达式实例

路径表达式	描述
//book/title | //book/price	选取所有 book 元素的 title 和 price 元素
//title | //price	选取文档中所有的 title 和 price 元素
/bookstore/book/title | //price	选取所有属于 bookstore 元素的 book 元素的 title 元素，以及文档中所有的 price 元素

在 XPath 表达式中可以使用常规的算术运算符、关系运算符、逻辑运算符等，常用的运算符如表 4-6 所示。

表 4-6 XPath 表达式的常用的运算符

运算符	描述	实例
|	计算两个结点集	//book | //cd，返回所有带有 book 和 cd 元素的结点集
+	加法	6 + 4，
-	减法	6 - 4
*	乘法	6 * 4
div	除法	8 div 4
=	等于	price=15

（续表）

运算符	描述	实例
!=	不等于	price!=15
<	小于	price<15
<=	小于或等于	price<=15
>	大于	price>15
>=	大于或等于	price>=15
or	或	price=15 or price=35
and	与	price>15 and price<35
mod	计算除法的余数	5 mod 2

下述代码用于实现任务【描述 4.D.2】，使用微软的 XML DOM 对象来载入 XML 文档，并使用 selectNodes()函数从 XML 文档选取结点，演示 XPath 的使用。

该任务描述将使用如下 XML 文档。

【描述 4.D.2】 bookstore3.xml

```
<?xml version="1.0" encoding="GB2312"?>
<bookstore>
    <book category="COOKING">
        <title lang="en">Everyday Italian</title>
        <author>Giada De Laurentiis</author>
        <year>2005</year>
        <price>30.00</price>
    </book>
    <book category="CHILDREN">
        <title lang="cn">唐老鸭</title>
        <author>美国迪斯尼</author>
        <year>1992</year>
        <price>29.00</price>
    </book>
    <book category="WEB">
        <title lang="cn">Web 编程基础</title>
        <author>iTeacher</author>
        <year>2003</year>
        <price>36.00</price>
    </book>
</bookstore>
```

下述代码演示了使用微软的 XML DOM 对象来载入 XML 文档，并使用 selectNodes()函数从 XML 文档选取结点的方法。

```
var xmlDoc=new ActiveXObject("Microsoft.XMLDOM");
xmlDoc.async="false";
```

```
xmlDoc.load("books.xml");
xmlDoc.selectNodes(路径表达式);
```

其中，“路径表达式”使用 XPath 表达式替代即可。

下面的代码选取并输出 bookstore 元素下所有的 book 结点。

【描述 4.D.2】 bookstore3.html

```
<html>
<body>
<script language="JavaScript" type="text/javascript">
//DOM 对象初始化
var xmlDoc=new ActiveXObject("Microsoft.XMLDOM");
xmlDoc.async="false";
//装载 xml 文件
xmlDoc.load("bookstore3.xml");
//使用 selectNodes()函数选择特定的结点
var nodes=xmlDoc.selectNodes("/bookstore/book");
//循环输出结点内容
for (var i=0;i<nodes.length ;i++ )
{
    document.write("<xmp>");
    document.write(nodes[i].xml);
    document.write("</xmp>");
}
</script>
</body>
</html>
```

通过 IE 浏览器，查看该 HTML 页面，效果如图 4-3 所示。

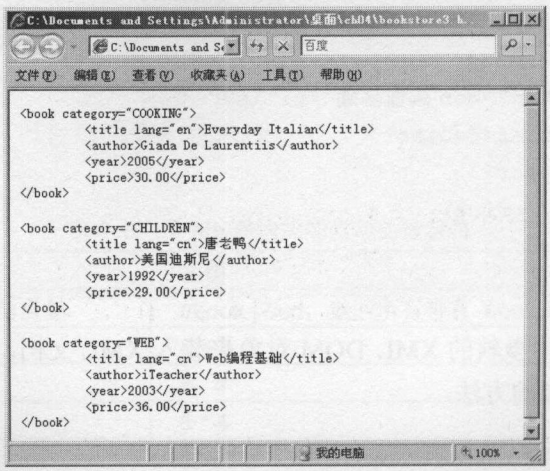


图 4-3 Xpath 的使用

对于下述代码片段，读者可参照注释自行调试加以理解。

```
//选取 bookstore 元素下第一个 book 结点
xmlDoc.selectNodes("/bookstore/book[0]");
//从所有的 price 结点选取文本
xmlDoc.selectNodes("/bookstore/book/price/text()");
//选取所有价格高于 35 的 price 结点
xmlDoc.selectNodes("/bookstore/book[price>35]/price");
//选取所有价格高于 35 的 title 结点
xmlDoc.selectNodes("/bookstore/book[price>35]/title");
```

注意　IE 5 和 IE 6 会把[0]作为第一个结点来执行，但是根据 W3C 的标准，应该使用[1]，这个问题在 IE 6 SP2 中被纠正了。

4.2.5　XPath 定位路径

一个 XPath 定位路径表达式将会返回一个结点集。对于定位路径既可以是绝对的也可以是相对的。绝对定位路径以一个斜线（/）开头，而相对定位路径没有。这两种情况下，定位路径由一个或多个定位步骤组成，每个步骤由一个斜线分隔，如下所示。

一个绝对定位路径：

```
/step/step/...
```

一个相对定位路径：

```
step/step/...
```

定位步骤从左到右，依次按顺序计算。每个步骤根据当前结点集的结点计算。如果定位路径是绝对的，那么当前结点集包含根结点。如果定位路径是相对的，那么当前结点集包含使用表达式的结点。

XPath 的定位步骤的语法如下：

```
轴名::结点测试[谓语]
```

其中：

- 轴（axis），定义某个相对于当前结点的结点集。
- 结点测试（node-test），识别某个轴内部的结点。
- 谓语（predicate），零或多个预测，以更深入地提炼所选的结点集。

如：

```
child::price[price=9.90]
```

其中，child 是轴，指示需要选取当前结点的所有子元素。XPath 中的轴名称如表 4-7 所示。

表 4-7　XPath 的轴名称

轴名称	描述
ancestor	选取当前结点的所有先辈
ancestor-or-self	选取当前结点的所有先辈以及当前结点本身
attribute	选取当前结点的所有属性
child	选取当前结点的所有子元素
descendant	选取当前结点的所有后代元素
descendant-or-self	选取当前结点的所有后代元素以及当前结点本身
following	选取文档中当前结点的结束标签之后的所有结点
namespace	选取当前结点的所有命名空间结点
parent	选取当前结点的父结点
preceding	选取文档中当前结点的开始标签之前的所有结点
preceding-sibling	选取当前结点之前的所有同级结点
self	选取当前结点

对于下述 XML 文档，表 4-8 列出了一些 XPath 定位路径表达式，以及表达式的作用。

```
<?xml version="1.0" encoding="GB2312"?>
<bookstore>
    <book lang="en">
        <title>RESTful Web Services</title>
        <price>29.00</price>
    </book>
    <book lang="zh">
        <title>Java 编程基础</title>
        <price>46.00</price>
    </book>
</bookstore>
```

表 4-8　XPath 定位路径表达式及其作用实例

表达式	描述
child::book	选取所有属于当前结点的子元素的 book 结点
attribute::lang	选取当前结点的 lang 属性
child::*	选取当前结点的所有子元素
attribute::*	选取当前结点的所有属性
child::text()	选取当前结点的所有文本子结点
child::node()	选取当前结点的所有子结点
descendant::book	选取当前结点的所有 book 后代
ancestor::book	选择当前结点的所有 book 先辈
ancestor-or-self::book	选取当前结点的所有 book 先辈以及当前结点
child::*/child::price	选取当前结点的孙子结点中的所有 price 元素
/	选择文档根

谓语用于过滤一个结点集为一个新的结点集。谓语要放置在方括号（[]）中。表 4-9 列出了一些带谓语的 XPath 定位路径表达式，以及表达式的作用。

表 4-9　带谓语的 XPath 定位路径表达式及其作用实例

表达式	描述
child::price[price=15.00]	选取当前结点的子结点中 price 元素等于 15.00 的所有元素
child::book[position()=1]	选择当前结点的第一个 book 子元素
child::book[position()=last()]	选择当前结点的最后一个 book 子元素
child::book[position()=last()-1]	选择当前结点的倒数第二个 book 子元素
child::book[position()<6]	选择当前结点的前五个 book 子元素
child::book[attribute::lang="en"]	选择当前结点的所有 lang 属性等于 en 的 book 子结点

定位路径也可以用缩写表示。最重要的缩写为 child::可以从一个定位步骤中省略。常用的缩写及其含义和示例如表 4-10 所示。

表 4-10　常用的缩写及含义

缩写	含义
none	child::
@	attribute::
.	self::node()
..	parent::node()
//	/descendant-or-self::node()

4.3　XSL 文档

XSL 文档包含一组模板规则和其他规则。模板规则拥有模式（pattern）以及模板（template），模式用来指定模板规则所适用的树形结构，而模板是用来在与此模式匹配时进行输出。当 XSL 处理程序使用 XSL 样式表来格式化 XML 文档时，它对 XML 文档树形结构进行扫描，依次浏览每个子树形结构。当读完 XML 文档中的每个树形结构时，处理程序就把它与样式表中每个模板规则的模式进行比较。当处理程序找到与模板规则的模式相匹配的树形结构时，它就输出此规则的模板。这个模板通常包括一些标记、新的数据和从原 XML 文档的树形结构中复制来的数据。

4.3.1　XSL 文档结构

XSL 文档本身是格式良好的 XML 文档，在 XSL 文件的开头要有和 XML 文件相同的声明，XSL 文档的标准格式如下：

```
<xsl:stylesheet version="1.0"xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
模板规则
输出模板
</xsl:stylesheet>
```

其中，<xsl:stylesheet>既是 XSL 的声明语句，也是根元素，必须位于文件的首部；xsl 是代表着 stylesheet 元素所属命名空间的前缀；xmlns:xsl="http://www.w3.org/1999/XSL/Transform"这一句主要用来说明该 XSL 样式表是使用 W3C 所制定的推荐标准。

注意 也可以使用<xsl:transform>作为 XSL 样式表的根元素，其作用与<xsl:stylesheet>时一致。

下述代码用于实现任务【描述 4.D.3】，基于给定的 XML，通过创建 XSL 文档，理解 XSL 的基本结构和显示效果。

该任务描述将使用如下 XML 文档，如无特殊说明，后续任务描述都使用该 XML 文档。

【描述 4.D.3】 student.xml

```
<?xml version="1.0" encoding="gb2312"?>
<class>
    <student ID="101">
        <name>李华</name>
        <sex>男</sex>
        <birthday>1988.12.12</birthday>
        <score>92</score>
        <skill>Java</skill>
    </student>
    <student ID="102">
        <name>王燕</name>
        <sex>女</sex>
        <birthday>1989.9.10</birthday>
        <score>89</score>
        <skill>Visual Basic</skill>
    </student>
    <student ID="103">
        <name>张磊</name>
        <sex>男</sex>
        <birthday>1987.9.9</birthday>
        <score>98</score>
        <skill>UML</skill>
    </student>
    <student ID="104">
        <name>杨菲</name>
        <sex>女</sex>
        <birthday>1989.6.16</birthday>
        <score>85</score>
        <skill>SQL Server</skill>
    </student>
```

```
</class>
```

创建 XSLT 文档 student.xsl 如下，XSLT 文件的后缀名为.xsl。

【描述 4.D.3】 student.xsl

```
<?xml version="1.0" encoding="gb2312"?>
<xsl:stylesheet version="1.0"
    xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
    <xsl:template match="/">
        <html>
            <head>
                <title>学生成绩单</title>
            </head>
            <body>
                <h2 align="center">学生成绩单</h2>
                <table border="2" cellpadding="0" align="center">
                    <tr bgcolor="#9acd32">
                        <th>姓名</th>
                        <th>性别</th>
                        <th>生日</th>
                        <th>成绩</th>
                    </tr>
                    <xsl:for-each select="/class/student">
                        <tr>
                            <td>
                                <xsl:value-of select="name"/>
                            </td>
                            <td>
                                <xsl:value-of select="sex"/>
                            </td>
                            <td>
                                <xsl:value-of select="birthday"/>
                            </td>
                            <td>
                                <xsl:value-of select="score"/>
                            </td>
                        </tr>
                    </xsl:for-each>
                </table>
            </body>
        </html>
    </xsl:template>
</xsl:stylesheet>
```

XSL 的规则是根据 XML 的规则制定的，通过分析该 XSL 文件，可以发现 XSL 文件的结构和 XML 文件的结构一致，一个 XSL 文件必须是 Well-Formed 的。另外，XSL 文件中所用的标记都是 W3C 定义好的，可以在 XSL 文件中插入一些 HTML 的标记，来帮助 XML 文件的显示，XSL 文件就是这些标记和 HTML 标记的组合，而这些 HTML 的标记也必须是 Well-Formed 的，如：必须要有正确的结束标记等。

如要在 XML 中调用 XSL 文件，使用 XSL 来展现 XML 文件，只要在 XML 文件声明语句后面加上一句：

```
<?xml-stylesheet type="text/xsl" href="要引用的样式表的路径及文件名"?>
```

和引用 CSS 是一样的，只不过是指定 type="text/xsl"。

对于 student.xml，只需在 XML 声明语句下面加入代码：

```
<?xml-stylesheet type="text/xsl" href="student.xsl"?>
```

通过 IE 浏览器查看 student.xml，效果如图 4-4 所示。

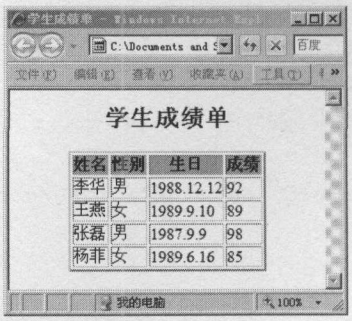


图 4-4　显示结果

在 XML 中引用 XSL 时应注意如下两点。

- 如果在声明部分引用了多个 XSL 样式表，则只有第一个样式表会生效，其余的都会被忽略掉。
- 本例中，引入 XSL 样式表时使用的是相对路径的方式，所以 xml 文档与其所要使用的 XSL 样式表文档必须位于同一路径下。
- 在 XSL 文件中，除 HTML 标记外，其他就是 XSL 自定义标记。

4.3.2　XSL 元素

模板（template）是 XSL 中最重要的概念之一。XSL 样式表由一个或多套被称为模板的规则组成，任何一个 XSL 文档至少包含一个模板。每个模板含有当某个指定的结点被匹配时所应用的规则，可以将模板看做一个方法，一个类，或者一个模块。它们可以被拼装组合，也可以单独成块，不同的模板控制不同的输出格式。

模板由两部分组成：匹配模式（match pattern）和执行。其中，模式用来定义 XML 文档中哪一个结点将被模板处理；执行则是用来定义输出的格式。两部分对应的语法中要用到的元素分别为<xsl:template>和<xsl:apply-templates>。

1. <xsl:template>元素

由<xsl:template>元素定义的模板规则是 XSL 样式表最重要的部分。每个模板规则是一个<xsl:template>元素，这些规则将特定的输出与特定的输入相关联。每个<xsl:template>元素都有一个 match 属性，用来指定要将此模板应用于输入文档的哪些结点，match 属性的属性值就是一个 XPath 表达式。

对于任务【描述 4.D.3】中的 student.xml 文档，下面的语句说明模板匹配所有的 name 元素：

```
<xsl:template match="name">
</xsl:template>
```

下面的语句写法说明模板匹配所有的 name 元素和所有的 student 元素：

```
<xsl:template match="(student|name)">
</xsl:template>
```

下面的语句写法说明模板匹配所有的父结点为 student 元素的 name 元素：

```
<xsl:template match="student//name">
</xsl:template>
```

下面的语句写法说明模板匹配根结点：

```
<xsl:template match="/">
</xsl:template>
```

<xsl:template>元素的内容是要运用的实际模板。模板中既可以包含需要显示在输出文档中的文本，同时也可以包含从 XML 文档提取数据的 XSL 指令。

注意　所有的 XSL 指令都放在 XSL 命名域中（即它们都是以 xsl: 开头），所以要区分元素（这些元素就是复制到输出的实际数据）和 XSL 指令是非常容易的。

例如，下面的代码是一个应用于输入树形结构根结点的模板：

```
<xsl:template match="/">
    <html>
        <head>
        </head>
        <body>
        </body>
    </html>
```

```
</xsl:template>
```

当 XSL 处理程序读取此输入文档时，它所看到的第一个结点就是根结点。下面的规则与此根结点相匹配，并告诉 XSL 处理程序发送此文本：

```
<html>
    <head>
    </head>
    <body>
    </body>
</html>
```

如果要在 XSL 样式表中使用上面的规则，那么输出就只限于上面的六个标记。这是由于在规则中没有任何指令告诉格式化程序沿树形结构逐级下移，以便寻找与样式表中的模板进一步的匹配。

下述代码用于实现任务【描述 4.D.4】，基于给定的 XML，通过创建 XSL 文档，演示<xsl:template>的使用。

【描述 4.D.4】 template.xsl

```
<?xml version="1.0" encoding="gb2312"?>
<xsl:stylesheet version="1.0"
    xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
    <xsl:template match="/">
        <html>
            <head>
                <title>学生成绩单</title>
            </head>
            <body>
                <h2 align="center">学生成绩单</h2>
                <table border="2" cellpadding="0" align="center">
                    <tr bgcolor="#9acd32">
                        <th>姓名</th>
                        <th>性别</th>
                        <th>生日</th>
                        <th>成绩</th>
                    </tr>
                    <tr>
                        <td/>
                        <td/>
                        <td/>
                        <td/>
                    </tr>
```

```
            </table>
        </body>
    </html>
</xsl:template>
</xsl:stylesheet>
```

上述 XSL 文档中，<xsl:template>元素定义了一个模板。而 match="/"属性则把此模板与 XML 源文档的根相联系。<xsl:template>元素内部的内容定义了写到输出的 HTML 代码。最后两行定义了模板的结尾及样式表的结尾。

在 student.xml 中引入该 XSL，并通过 IE 浏览器查看 student.xml，效果如图 4-5 所示。

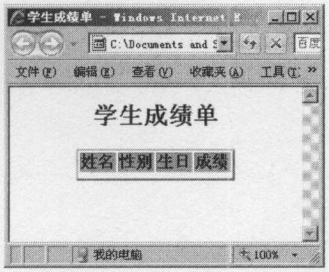


图 4-5　显示结果

2. <xsl:value-of>元素

任务【描述 4.D.4】没有将 XML 文档的数据复制到输出，使用<xsl:value-of>元素可用来取出 XML 文件中被选择的元素或属性的内容，语法格式如下：

```
<xsl:value-of  select ="匹配模式"/>
```

在<xsl:value-of>中，输出指定结点的取值，用 select 属性进行限定；如果要显示属性的值，必须在属性名之前加“@”符号作为前缀。而匹配模式是一个 XPath 表达式。

注意　<xsl:value-of>元素是一个空元素，在结束前，应有"/"。

下述代码用于实现任务【描述 4.D.5】，基于给定的 XML，通过创建 XSL 文档，演示<xsl:value-of>的使用。

【描述 4.D.5】 value-of.xsl

```
<?xml version="1.0" encoding="gb2312"?>
<xsl:stylesheet version="1.0"
    xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
    <xsl:template match="/">
        <html>
            <head>
                <title>学生成绩单</title>
            </head>
```

```
        <body>
            <h2 align="center">学生成绩单</h2>
            <table border="2" cellpadding="0" align="center">
                <tr bgcolor="#9acd32">
                    <th>姓名</th>
                    <th>ID</th>
                    <th>性别</th>
                    <th>生日</th>
                    <th>成绩</th>
                </tr>
                <tr>
                    <td>
                        <xsl:value-of select="class/student/name"/>
                    </td>
                    <td>
                        <xsl:value-of select="class/student/@ID"/>
                    </td>
                    <td>
                        <xsl:value-of select="class/student/sex"/>
                    </td>
                    <td>
                        <xsl:value-of select="class/student/birthday"/>
                    </td>
                    <td>
                        <xsl:value-of select="class/student/score"/>
                    </td>
                </tr>
            </table>
        </body>
    </html>
  </xsl:template>
</xsl:stylesheet>
```

上述 XSL 文档中，在<xsl:value-of>元素中使用 select 属性进行元素定位，读者需特别注意对 student 元素的 ID 属性进行定位的 XPath 表达式。

在 student.xml 中引入该 XSL，并通过 IE 浏览器查看，其效果如图 4-6 所示，在此示例中仅有一行数据从 XML 文档被复制到输出结果。

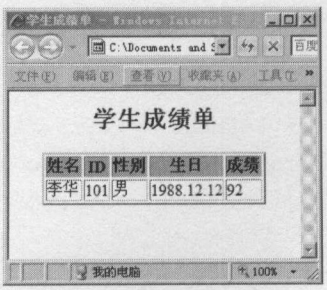


图 4-6　显示结果

假如将 XSL 文档写成如下形式（注意加粗部分）。

【描述 4.D.5】 value-of_2.xsl

```
<?xml version="1.0" encoding="gb2312"?>
<xsl:stylesheet version="1.0"
    xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
    <xsl:template match="class/student">
        <html>
            <head>
                <title>学生成绩单</title>
            </head>
            <body>
                <h2 align="center">学生成绩单</h2>
                <table border="2" cellpadding="0" align="center">
                    <tr bgcolor="#9acd32">
                        <th>姓名</th>
                        <th>ID</th>
                        <th>性别</th>
                        <th>生日</th>
                        <th>成绩</th>
                    </tr>
                    <tr>
                        <td>
                            <xsl:value-of select="./name"/>
                        </td>
                        <td>
                            <xsl:value-of select="./@ID"/>
                        </td>
                        <td>
                            <xsl:value-of select="./sex"/>
                        </td>
                        <td>
                            <xsl:value-of select="birthday"/>
                        </td>
                        <td>
                            <xsl:value-of select="score"/>
                        </td>
                    </tr>
                </table>
            </body>
        </html>
    </xsl:template>
</xsl:stylesheet>
```

本样式表中，模板的匹配位置是 class 元素下的所有名为 student 的子元素；对于<xsl:value-of>元素中的 select 属性的内容——“.”指的是当前位置，即模板的匹配位置 class/student，匹配模式“./@ID”指的就是所有 student 元素的 ID 属性，最终输出的内容就是这些 student 元素的 ID 属性的内容。

在 student.xml 中引入该 XSL，并通过 IE 浏览器查看，其效果如图 4-7 所示。

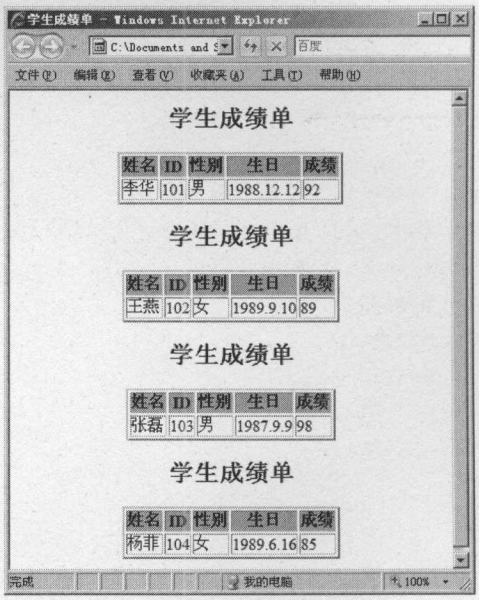


图 4-7　显示结果

3. <xsl:for-each>元素

如果有多个结点可供选择的话，<xsl:value-of>元素只能获得第一个结点中的内容。若想循环选择多条数据，可以使用<xsl:for-each>元素。其语法格式如下：

```
<xsl:for-each select="匹配模式">
    <!--输出内容-->
</xsl:for-each>
```

其中匹配模式是一个 XPath 表达式。

下述代码用于实现任务【描述 4.D.6】，基于给定的 XML，通过创建 XSL 文档，演示<xsl:for-each>的使用。

【描述 4.D.6】 for-each.xsl

```
<?xml version="1.0" encoding="gb2312"?>
<xsl:stylesheet version="1.0"
    xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
    <xsl:template match="/">
        <html>
```

```
        <head>
            <title>学生成绩单</title>
        </head>
        <body>
            <h2 align="center">学生成绩单</h2>
            <table border="2" cellpadding="0" align="center">
                <tr bgcolor="#9acd32">
                    <th>姓名</th>
                    <th>性别</th>
                    <th>生日</th>
                    <th>成绩</th>
                </tr>
                <xsl:for-each select="/class/student[score > 90]">
                    <tr>
                        <td>
                            <xsl:value-of select="name"/>
                        </td>
                        <td>
                            <xsl:value-of select="sex"/>
                        </td>
                        <td>
                            <xsl:value-of select="birthday"/>
                        </td>
                        <td>
                            <xsl:value-of select="score"/>
                        </td>
                    </tr>
                </xsl:for-each>
            </table>
        </body>
    </html>
</xsl:template>
</xsl:stylesheet>
```

本样式表中，在<xsl:for-each>元素的 select 属性值中加入判定语句“score>90”用于过滤从 XML 文件输出的结果。

在 student.xml 中引入该 XSL，并通过 IE 浏览器查看，其效果如图 4-8 所示。



图 4-8　显示结果

4. <xsl:sort>元素

如果要对结果进行排序，只要在 XSL 文件中的<xsl:for-each>元素内部添加一个<xsl:sort>元素即可。其语

法格式如下：

```
<xsl:sort select="匹配模式" order="ascending|descending"
        case-order="upper-first|lower-first" data-type="text|number|qname"/>
```

其中：

- data-type 表示被排序的数据类型，可以是数字、文本或用户自定义，默认是文本。
- select 属性指示需要排序的 XML 元素。
- order 属性指示排序方式为升序或降序，默认为 ascending。
- case-order 属性设置为 upper-first 或 lower-first 两个值当中的一个，以指定大写字母是排在小写字母之前，还是反过来。默认状况依赖于语言。

下述代码用于实现任务【描述 4.D.7】，基于给定的 XML，通过创建 XSL 文档，演示<xsl:sort>的使用。

【描述 4.D.7】 sort.xsl

```
<?xml version="1.0" encoding="gb2312"?>
<xsl:stylesheet version="1.0"
    xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
    <xsl:template match="/">
        <html>
            <head>
                <title>学生成绩单</title>
            </head>
            <body>
                <h2 align="center">学生成绩单</h2>
                <table border="2" cellpadding="0" align="center">
                    <tr bgcolor="#9acd32">
                        <th>姓名</th>
                        <th>性别</th>
                        <th>生日</th>
                        <th>成绩</th>
                    </tr>
                    <xsl:for-each select="/class/student">
                    <xsl:sort select="score" order="descending"/>
                        <tr>
                            <td>
                                <xsl:value-of select="name"/>
                            </td>
                            <td>
                                <xsl:value-of select="sex"/>
```

```
                        </td>
                        <td>
                            <xsl:value-of select="birthday"/>
                        </td>
                        <td>
                            <xsl:value-of select="score"/>
                        </td>
                    </tr>
                </xsl:for-each>
            </table>
        </body>
    </html>
</xsl:template>
</xsl:stylesheet>
```

本样式表中，通过在<xsl:sort>元素中设置相关属性使 XML 输出内容按照 score 列（select="score"）降序（order="descending"）输出。

在 student.xml 中引入该 XSL，并通过 IE 浏览器查看，其效果如图 4-9 所示。

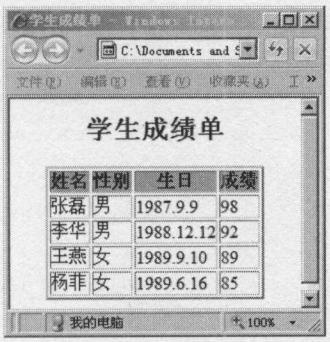


图 4-9　显示结果

5. <xsl:if>元素

<xsl:if>类似普通程序语言的 if 条件语句，允许设定结点满足某个条件时被模板处理。其语法格式如下：

```
<xsl:if test="匹配模式">
    <!--输出内容-->
</xsl:if>
```

其中，匹配模式用于进行元素匹配，在<xsl:if>元素中，通常可以使用如下三种匹配。

- 用元素名称作为匹配条件。
- 用元素内容作为匹配条件。
- 用元素的属性值作为匹配条件。

下述代码用于实现任务【描述 4.D.8】，基于给定的 XML，通过创建 XSL 文档，演示<xsl:if>的使用。

【描述 4.D.8】 if.xsl

```
<?xml version="1.0" encoding="gb2312"?>
<xsl:stylesheet version="1.0"
    xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
    <xsl:template match="/">
        <html>
            <head>
                <title>学生成绩单</title>
            </head>
            <body>
                <h2 align="center">学生成绩单</h2>
                <table border="2" cellpadding="0" align="center">
                    <tr bgcolor="#9acd32">
                        <th>姓名</th>
                        <th>性别</th>
                        <th>生日</th>
                        <th>成绩</th>
                    </tr>
                    <xsl:for-each select="/class/student">
                        <xsl:if test="score>90">
                            <tr>
                                <td>
                                    <xsl:value-of select="name"/>
                                </td>
                                <td>
                                    <xsl:value-of select="sex"/>
                                </td>
                                <td>
                                    <xsl:value-of select="birthday"/>
                                </td>
                                <xsl:if test="score">
                                    <td bgcolor="yellow">
                                        <xsl:value-of select="score"/>
                                    </td>
                                </xsl:if>
                            </tr>
                        </xsl:if>
                    </xsl:for-each>
                </table>
```

```
            </body>
        </html>
    </xsl:template>
</xsl:stylesheet>
```

本样式表中，显示了成绩大于 90（test="score>90"）的学生的成绩单，并且对学生的 score 列以黄色背景（test="score"）显示。

在 student.xml 中引入该 XSL，并通过 IE 浏览器查看，其效果如图 4-10 所示。

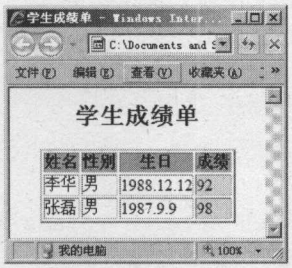


图 4-10　显示结果

6. <xsl:choose>元素

<xsl:if>元素没有 else 的属性。如果需要进行多项选择，可以使用<xsl:choose>、<xsl:when>、<xsl:otherwise>系列流程控制语句。<xsl:choose>、<xsl:when>、<xsl:otherwise>类似于 Java 中的 switch 语句，其语法格式如下：

```
<xsl:choose>
    <xsl: when test="匹配模式">
        <!--输出内容-->
    </xsl:when>
    <xsl:when test="匹配模式">
        <!--输出内容-->
    </xsl:when>
        ... ...
    <xsl:otherwise>
        <!--输出内容-->
    </xsl:otherwise>
</xsl:choose>
```

下述代码用于实现任务【描述 4.D.9】，基于给定的 XML，通过创建 XSL 文档，演示<xsl:choose>的使用。

【描述 4.D.9】 choose.xsl

```
<?xml version="1.0" encoding="gb2312"?>
<xsl:stylesheet version="1.0"
    xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
```

```
<xsl:template match="/">
    <html>
        <head>
            <title>学生成绩单</title>
        </head>
        <body>
            <h2 align="center">学生成绩单</h2>
            <table border="2" cellpadding="0" align="center">
                <tr bgcolor="#9acd32">
                    <th>姓名</th>
                    <th>性别</th>
                    <th>生日</th>
                    <th>成绩</th>
                </tr>
                <xsl:for-each select="/class/student">
                    <tr>
                        <td>
                            <xsl:value-of select="name"/>
                        </td>
                        <td>
                            <xsl:value-of select="sex"/>
                        </td>
                        <td>
                            <xsl:value-of select="birthday"/>
                        </td>
                        <xsl:choose>
                            <xsl:when test="score > 90">
                                <td bgcolor="#ff00ff">
                                    <xsl:value-of select="score"/>
                                </td>
                            </xsl:when>
                            <xsl:when test="score > 85">
                                <td bgcolor="yellow">
                                    <xsl:value-of select="score"/>
                                </td>
                            </xsl:when>
                            <xsl:otherwise>
                                <td>
                                    <xsl:value-of select="score"/>
                                </td>
                            </xsl:otherwise>
                        </xsl:choose>
                    </tr>
```

```
                </xsl:for-each>
            </table>
        </body>
    </html>
</xsl:template>
</xsl:stylesheet>
```

上述样式表中，对成绩大于 90 分（test="score>90"）的采用粉色背景显示，成绩在 85 分到 90 分之间（test="score>85"）的采用黄色背景显示，其他采用默认颜色背景。

在 student.xml 中引入该 XSL，并通过 IE 浏览器查看，其效果如图 4-11 所示。



图 4-11　显示结果

7. <xsl:apply-templates>元素

<xsl:apply-templates>元素用来指示处理器发现一个合适的模板，并且在每个选择的元素上执行指定的任务，可以将其理解为程序中的子函数调用。其语法格式如下：

```
<xsl:apply-templates select ="匹配模式">
</xsl:apply-templates>
```

其中，select 属性用来定义确切的结点名称，是可选的，默认值是“node()”，意味着对当前结点的所有子结点执行该模板。在给 select 属性指定属性值时也同样要用到前面介绍的 XPath 相关知识。另外要注意，xsl:apply-templates 总是包含在 xsl:template 元素中，格式如下：

```
<xsl:template match="/">
    <xsl:apply-templates select="student"/>
</xsl:template>
```

上面这段代码说明模板匹配整个文档（从根结点开始匹配），具体执行时处理根结点下所有 student 元素。而对于下列代码：

```
<xsl:template match="student">
    <p><xsl:apply-templates/></p>
</xsl:template>
```

则表示模板匹配 student 结点，即处理的是所有 student 下的子元素。

下述代码用于实现任务【描述 4.D.10】，基于给定的 XML，通过创建 XSL 文档，演示<xsl:apply-templates>的使用。

【描述 4.D.10】 apply–templates.xsl

```
<?xml version="1.0" encoding="gb2312"?>
<xsl:stylesheet version="1.0"
    xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
    <xsl:template match="/">
        <html>
            <head>
                <title>学生成绩单</title>
            </head>
            <body>
                <xsl:apply-templates/>
            </body>
        </html>
    </xsl:template>
    <xsl:template match="student">
        <p>
            <xsl:apply-templates select="name"/>
            <xsl:apply-templates select="score"/>
        </p>
    </xsl:template>
    <xsl:template match="name">
            Name: <span style="color:#ff0000">
            <xsl:value-of select="."/>
        </span>
        <br/>
    </xsl:template>
    <xsl:template match="score">
            Score: <span style="color:#00ff00">
            <xsl:value-of select="."/>
        </span>
        <br/>
    </xsl:template>
</xsl:stylesheet>
```

在 student.xml 中引入该 XSL，并通过 IE 浏览器查看，其效果如图 4-12 所示。

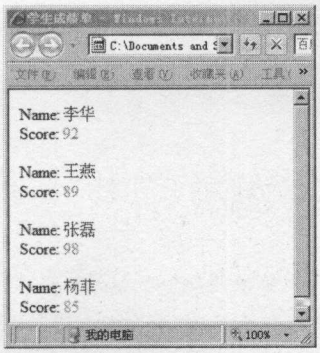


图 4–12　显示结果

4.4　多样式表

在 XSLT 中，可以使用<xsl:import>和<xsl:include>元素来合并多个样式表。

4.4.1　<xsl:import>元素

<xsl:import>元素为顶级元素，用于把一个样式表中的内容导入另一个样式表中。其语法格式如下：

```
<xsl:import href="location">
```

其中：

- href 属性提供导入的样式表的 URI。
- <xsl:import>元素必须是出现在 stylesheet 元素之后的一个顶级元素。

例如，下面的代码片段使用<xsl:import>元素导入 first.xsl 和 second.xsl 样式表。

```
<xsl:stylesheet xmlns:xsl="http://www.w3.org/TR/WD-xsl">
<xsl:import href="first.xsl"/>
<xsl:import href="second.xsl"/>
……
</xsl:stylesheet>
```

在使用<xsl:import>时需要注意，如果导入的样式表中的规则与执行导入的样式表中的规则发生冲突，那么执行导入的样式表中的规则优先；如果不同的被导入样式表中的两个规则发生冲突，那么最后一个被导入的样式表中的规则优先。

下述代码用于实现任务【描述 4.D.11】，创建 XSL 文档，演示<xsl:import>元素的使用。

新建一个 XSL 样式表文件（first.xsl），其内容如下所示。

【描述 4.D.11】 first.xsl

```
<?xml version="1.0" encoding="gb2312"?>
<xsl:stylesheet version="1.0"
    xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
    <xsl:template match="class/student">
        <p>
            <xsl:apply-templates select="name"/>
            <xsl:apply-templates select="score"/>
        </p>
    </xsl:template>
    <xsl:template match="name">
            Name: <span style="color:#ff0000">
            <xsl:value-of select="."/>
        </span>
        <br/>
    </xsl:template>
    <xsl:template match="score">
            Score: <span style="color:#00ff00">
            <xsl:value-of select="."/>
        </span>
        <br/>
    </xsl:template>
</xsl:stylesheet>
```

再新建一个 XSL 样式表文件（import.xsl），并在此样式表中导入 first.xsl，其内容如下所示。

【描述 4.D.11】 import.xsl

```
<?xml version="1.0" encoding="gb2312"?>
<xsl:stylesheet version="1.0"
    xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
    <xsl:import href="first.xsl"/>
    <xsl:template match="/">
        <xsl:apply-templates select="class/student"/>
    </xsl:template>
</xsl:stylesheet>
```

第三行代码导入了名为 first.xsl 的样式表文件中的内容。此样式表文件将应用所导入的样式表文件中定义的模板规则。

在 student.xml 中引入该 XSL，并通过 IE 浏览器查看，其效果与图 4-12 一致。

4.4.2　<xsl:include>元素

<xsl:include>元素也是顶级元素，其语法类似于<xsl:import>元素。它将另一个样式表导入到当前的样式表中所指定的位置处，其 href 属性提供要导入的样式表的 URI。与<xsl:import>元素所不同的是，<xsl:include>元素所包括的规则与执行包括的样式表中的规则具有同样的优先级。

<xsl:include>元素可放在顶级处于最后一个<xsl:import>元素之后的任何地方。

下述代码用于实现任务【描述 4.D.12】，创建 XSL 文档，演示<xsl:include>元素的使用。

新建一个 XSL 样式表文件（include.xsl），内容如下所示。

【描述 4.D.12】 include.xsl

```
<?xml version="1.0" encoding="gb2312"?>
<xsl:stylesheet version="1.0"
    xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
    <xsl:template match="/">
        <html>
            <head>
                <title>学生成绩单</title>
            </head>
            <body>
                <h2 align="center">学生成绩单</h2>
                <table border="2" cellpadding="0" align="center">
                    <tr bgcolor="#9acd32">
                        <th>姓名</th>
                        <th>性别</th>
                        <th>生日</th>
                        <th>成绩</th>
                    </tr>
                    <xsl:for-each select="/class/student[score &gt; 90]">
                    <tr>
                        <xsl:apply-templates select="name"/>
                        <xsl:apply-templates select="sex"/>
                        <xsl:apply-templates select="birthday"/>
                        <xsl:apply-templates select="score"/>
                    </tr>
                    </xsl:for-each>
                </table>
            </body>
        </html>
    </xsl:template>
    <xsl:template match="name">
```

```
        <td>
            <xsl:value-of select="."/>
        </td>
    </xsl:template>
    <xsl:template match="sex">
        <td>
            <xsl:value-of select="."/>
        </td>
    </xsl:template>
    <xsl:include href="second.xsl"/>
</xsl:stylesheet>
```

其包含了名为 second.xsl 的样式表，其内容如下所示。

【描述 4.D.12】 second.xsl

```
<?xml version="1.0" encoding="gb2312"?>
<xsl:stylesheet version="1.0"
    xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
    <xsl:template match="birthday">
        <td>
            <xsl:value-of select="."/>
        </td>
    </xsl:template>
    <xsl:template match="score">
        <td bgcolor="#ff00ff">
            <xsl:value-of select="."/>
        </td>
    </xsl:template>
</xsl:stylesheet>
```

在 student.xml 中引入该 include.xsl，并通过 IE 浏览器查看，其效果如图 4-13 所示。



图 4-13 显示结果

小结

通过本章的学习，学生应该能够学会：

- CSS 可以格式化 XML 文档，能够很好地控制输出的样式，但只能静态控制文档的样式或外观。
- XSL 是遵循 XML 的规范来制定的。
- XSL 实际上包含三种语言：XSLT、XPath 和 XSLF。
- 从狭义上理解，XSL 和 XSLT 是一样的。
- 对 XPath 的理解是很多高级 XML 应用的基础。
- XPath 可以用于 XSLT 样式表内来定位 XML 树中的某个位置。
- XPath 表达式由左至右读取，路径中的各个点由一个前向斜杠（/）分隔。
- XPath 主要有四种数据类型：结点集、布尔类型、数值类型、字符串。
- XPath 中，结点之间的关系有父、子、同胞、先辈、后代。
- XPath 使用路径表达式在 XML 文档中选取结点，并可以使用谓语查找某个特定的结点或者包含某个值的结点。
- 如果需要在转换过程中进行运算，可以在 XPath 表达式中使用数学运算符或函数，在 XPath 表达式中可以使用常规的算术运算符、关系运算符、逻辑运算符等。
- XSL 文档本身是格式良好的 XML 文档。
- XSL 模板由两部分组成：匹配模式和执行。
- 在 XSLT 中，可以使用<xsl:import>和<xsl:include>元素来合并多个样式表。

练习

1. 由_____元素定义的模板规则是 XSL 样式表的最重要的部分。
   A. <xsl:value-of>
   B. <xsl:template>
   C. <xsl:apply-templates>
   D. <xsl:choose>
2. <xsl:sort>元素一般作为<xsl:apply-templates>或_____的子元素出现。
   A. <xsl:value-of>
   B. <xsl:template>
   C. <xsl:apply-templates>
   D. <xsl:for-each>
3. 下列属于 XPath 的数据类型的是_____。（多选）
   A. 布尔类型

B. 字符串

C. 结点集

D. 数值类型

4. CSS 的定义由三个部分构成：_____、_____和_____。

5. XSL 实际上包含三种语言：_____、_____和_____。

6. XPath 中，结点之间的关系有父、_____、同胞、_____、后代。

7. 对于如下 XML 文档：

```
<?xml version="1.0" encoding="GB2312"?>
<bookstore>
    <book category="COOKING">
        <title lang="en">Everyday Italian</title>
        <author>Giada De Laurentiis</author>
        <year>2005</year>
        <price>30.00</price>
    </book>
    <book category="CHILDREN">
        <title lang="cn">唐老鸭</title>
        <author>美国迪斯尼</author>
        <year>1992</year>
        <price>29.00</price>
    </book>
</bookstore>
```

选取 bookstore 元素下所有的 book 子元素使用的 XPath 表达式是：_____。

选取所有价格高于 35 的 price 结点的 XPath 表达式是：_____。

选取 bookstore 元素下最后一个 book 结点的 XPath 表达式是：_____。

选取所有 book 元素的 title 和 price 元素的 XPath 表达式是：_____。

8. 对于下述 XML 文档：

```
<?xml version="1.0" encoding="GB2312"?>
<Orders>
    <Order orderID="T001" orderDate="2010-1-20">
        <name>遥控车</name>
        <number>300</number>
        <city>北京</city>
        <zip>100000</zip>
    </Order>
    <Order orderID="A002" orderDate="2010-3-20">
        <name>订书机</name>
        <number>180</number>
        <city>青岛</city>
```

```
        <zip>266000</zip>
    </Order>
</Orders>
```

根据要求编写并在 XML 文档上应用对应的的 XSL 样式表：

- 以表格的形式显示 XML 文档中存储的所有数据。
- 显示 orderID 为 A002 的订单信息。
- 显示北京的订单信息。

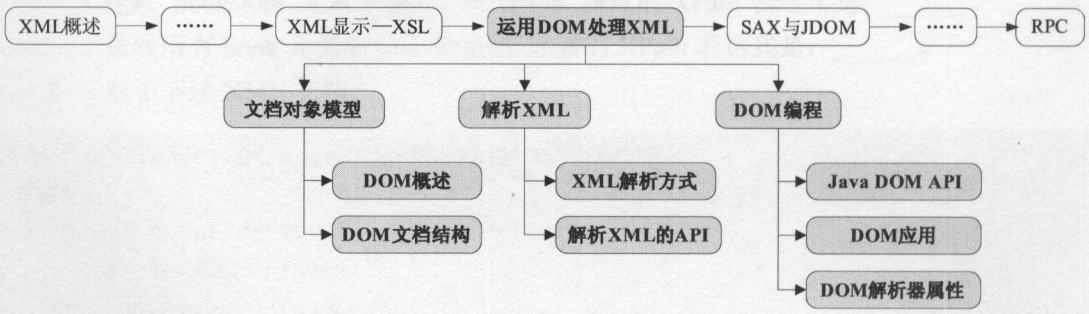
9. 简要描述 XSL 解析 XML 文档的过程。

第 5 章　运用 DOM 处理 XML

本章目标

- 了解 DOM 的概念
- 了解 DOM 处理 XML 的优势
- 掌握 DOM 的文档结构
- 理解 DOM 的常用结点
- 了解常用的 XML 解析器及其特点
- 了解 JAXP 的基本结构
- 掌握 JAXP 处理 XML 的基本步骤
- 掌握常用的 DOM API
- 掌握基于 DOM 的增、删、改、查操作
- 了解 DOM 解析器各属性的作用

学习导航



任务描述

【描述 5.D.1】

使用 JAXP 完成处理 XML 的一般过程。

【描述 5.D.2】

通过载入给定 XML 文档，遍历并显示 XML 文档中的全部元素。

【描述 5.D.3】

创建 Java 程序，用于实现对任意给定的 XML 文档进行遍历。

【描述 5.D.4】

创建一个 XML 文档的 DOM 解析树，并将其保存成 XML 文档。

【描述 5.D.5】

基于任务【描述 5.D.4】创建的 XML 文档，实现向该 XML 文档插入学生信息及其子元素。

【描述 5.D.6】

创建验证程序，通过带 DOCTYPE 声明的 XML 文件检查解析器的验证情况。

5.1 文档对象模型

XML 为数据交换提供了一种与平台无关的语言，W3C DOM 规范提供了一种与浏览器、平台、语言皆无关的接口，使得程序设计人员能够用任何编程语言访问 XML 数据。

5.1.1 概述

XML DOM 是 XML Document Object Model 的缩写，即 XML 文档对象模型。XML DOM 定义了访问和处理 XML 文档的标准方法。DOM 是以层次结构组织在一起的结点或信息片断的集合，允许开发人员在树状的数据结构中导航寻找特定信息。分析时通常需要加载整个文档并构造层次结构，然后才能进行相关工作。

DOM 提供了一套 API，为 XML 文档处理提供了标准的功能定义，通过该接口，程序设计人员可以对文档中的数据、文档的结构进行各种操作。DOM 提供的对象和方法可以和任何编程语言（Java、C++、VB）一起使用，也可以与 VBScript、JavaScript 等脚本语言一起使用。

使用 DOM 处理 XML 有以下几点优越性。

- DOM 能够保证 XML 文档的语法正确和格式正规。

  由 DOM 将文本文件转化为抽象的结点树表示，因此能够完全避免无结束标记和不正确的标记嵌套等问题。使用 DOM 操作 XML 文档时，开发人员不必担心文档的文本表示，只需要关注父子关系和相关的信息。另外，DOM 能够避免文档中不正确的父子关系。例如，一个 Attr 对象永远也不能成为另一个 Attr 对象的父对象。
- DOM 能够从语法中提取内容。

  由 DOM 创建的结点树是 XML 文件内容的逻辑表示，它显示了文件提供的信息，以及它们之间的关系，而不受限于 XML 语法。例如，结点树蕴含的信息可以用于更新关系数据库，或者创建 HTML 页面，开发人员不必纠缠于 XML 语法规范。
- DOM 能够简化内部文档操作。

  使用 DOM 比使用传统的文件操作机制更加简单。DOM 提供了一套 API，通过该标准，程序设计人员可以从文档中读取、搜索、修改、增加和删除数据，操纵 XML 文档的内容和结构。
- DOM 能够贴切地反映典型的层次数据库和关系数据库的结构。

  DOM 表示数据元素关系的方式非常类似于现代的层次型和关系型数据库表示信息的方法。这使得利用 DOM 在数据库和 XML 文件之间移动信息变得相当简单。

5.1.2 DOM 文档结构

在 DOM 中，一般将 XML 逻辑结构描述成树。DOM 通过解析 XML 文档，为 XML 文档在逻辑上建立一个树模型（DOM 树），树的结点是一个个的对象。所以通过操作这棵树及其中的对象就可以完成对 XML 文档的操作，为处理文档的所有方面提供了一个完美的概念

性框架。

如：对于如下 XML 文档，其 DOM 树如图 5-1 所示。

```
<?xml version="1.0" encoding="GB2312"?>
<!-- DOM 示例-->
<bookstore>
    <book>
        <title lang="en">RESTful Web Services</title>
        <price>29.00</price>
    </book>
    <book>
        <title lang="zh">Java 编程基础</title>
        <price>46.00</price>
    </book>
</bookstore>
```

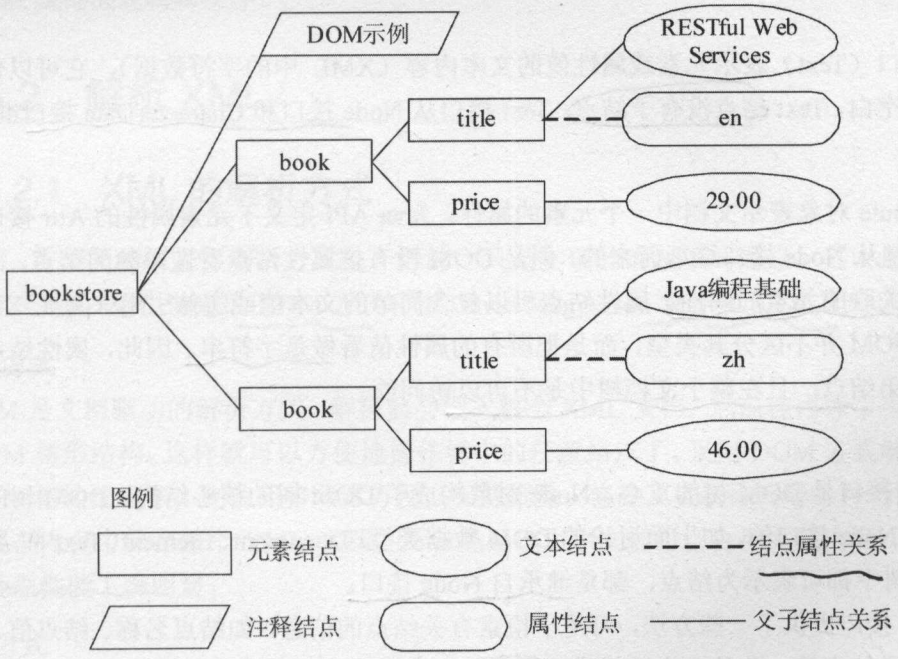


图 5-1　DOM 树

W3C 将 DOM 分成层而不是版本。DOM 的 Level1（核心）表示文档中内容的层次；Level2 提供了一套核心接口和模块，以改进 Level1 中所表示的一般结构。DOM 的 Level1 以 Node 对象的层次结构来表示文档，这些结点中的任何一个都可以有零个或更多不同类型的子结点，并且可以在其下形成子树。如图 5-1 的 DOM 树结构中，可把每个 XML 标记（和它的值）看做是一个结点（Node）对象。结点对象是 DOM 树的基本对象，XML 中共有 12 种结点类型，其中最常见的结点类型有文档、元素、属性、文本和注释 5 种。

注意 下述正文中所提及的接口基于 JAXP 工具包，其提供了处理 XML 文档中不同类型元素的多种接口。关于 JAXP 将在后续小节中介绍。

1. 文档

*XML 树中的所有结点的容器，也称为文档根。Document 接口指定 DOM 树的顶部结点，*并且包含完整的文档（XML 或 HTML）。它提供了对文档元素和数据的访问。由于所有其他结点必须在文档内，所以此接口包含创建其他结点的方法，以及操作文档结构和属性的方法，Document 接口继承自 Node 接口。

2. 元素

元素是（Element 接口）XML 的基本构件，元素可以包含其他元素、文本结点或两者的组合用来作为其子结点。Element 接口从 Node 接口继承而来，提供了一些方法用来设置或检索元素的属性，是唯一允许带有属性的结点。上述 XML 文档中的 book、title、price 都是 DOM 文档的元素。

3. 文本

该接口（Text）表示元素或属性值的文本内容（XML 中的字符数据），它可以包含信息也可以是空白，Text 结点没有子结点。Text 接口从 Node 接口和 CharacterData 接口继承而来。

4. 属性

Attribute 对象表示文档中一个元素的属性。Java API 定义了元素属性的 Attr 接口。虽然 Attr 接口是从 Node 接口继承而来的，但是 DOM 没有把属性结点看做单独的结点，而是作为与结点相关联的元素的属性。属性结点可以包含简单的文本值或实体引用（与在 XML 中一样），但 DOM 并不区分其类型，而是把所有的属性值看做是字符串。因此，属性结点没有父结点或兄弟结点，且在整个文档树中是不可以访问的。

5. 结点

Node 接口是 DOM 树的重心，Node 对象构成了 DOM 树的核心结构。DOM 树的整体架构是基于 Node 接口的，如上面讨论的 DOM 数据类型：Document、Element、Text 和 Attribute，在 DOM 树中都可表示为结点，都是继承自 Node 接口。

Node 接口提供了一些方法，可用于检索有关结点的信息，如结点名称、结点值、结点属性、相关子结点等。此外，它还提供了一种机制来获取和操作有关子结点的信息。

6. NodeList

NodeList 接口表示一个结点对象的集合。该接口提供一种便利机制，可以把树状文档结点转化为熟知的列表进行访问。通过该接口，可以循环某个结点集合，并且对文档结构所做的任何改变（如添加或删除结点），在结点列表中会立即反映出来。该结构对于 DOM 树的操作尤其有用。

在 DOM 结构树中，不同对象是相互关联的，这种结点关系可以是子结点、父结点、兄弟结点等，该层次如图 5-2 所示。

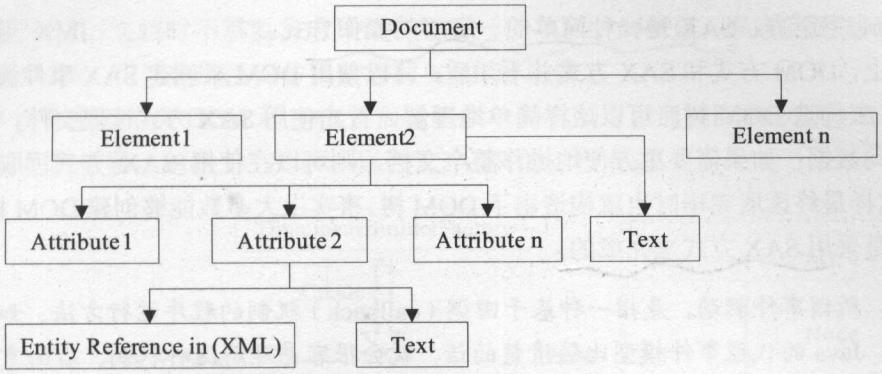


图 5-2　DOM 对象的层次

从图 5-2 可以看出，Document 对象必须位于层次的顶部，文档由 Element 对象组成，文档的实际数据或文本内容存储为 Text 对象，属性用于提供有关元素的附加信息，属性值也表示为文本对象。需要重点强调的是，DOM 结构中的对象不能任意放置，元素的嵌套方式要符合 DOM 规范的逻辑和次序。

5.2　解析 XML

5.2.1　XML 的解析方式

XML 在不同的语言里解析方式是一样的，只是实现的语法不同而已。其中最主要的两种解析方式是由 W3C 制定的 DOM 方式和由 David Megginson 领导的 SAX 方式。

1. DOM 方式

DOM 是文档驱动的解析方式。解析器会读入整个 XML 文档，然后在内存中构造一棵完整的 DOM 树形结构，这样就可以方便地操作树中的任意结点了。采用 DOM 方式解析 XML，一旦 DOM 树构造完毕，则访问将非常方便，支持删除、修改、重新排列等各种功能。但是，因为需要在内存中保存整个 XML 文档，所以当处理大型文档时会占用大量内存，并且加载速度和处理性能下降明显。

2. SAX 方式

SAX（Simple API for XML）是事件驱动的解析方式。当解析器发现元素开始或元素结束、文本、文档的开始或结束等标记时，会触发相应的事件，开发者可以通过编写响应这些事件的代码来保存数据。这种处理机制非常类似于流媒体的处理方式，分析能够立即开始，而不是等待所有的数据都被处理完毕后再开始。而且，由于应用程序只是在读取数据时检查数据，因此不需要将数据存储在内存中。这对于大型文档来说是一个巨大的优点。实际上，应用程序甚至不必解析整个文档，可以在某个条件得到满足时停止解析。

从上述对 SAX 和 DOM 的分析可以看出，实际上 SAX 和 DOM 不是一个操作层次上的概念，SAX 是在读取 XML 文件时即时进行操作，而 DOM 是在读取完毕后再进行操作，正

如 SAX 的名字所言，SAX 是一种简单的、底层的操作方式。

实际上，DOM 方式和 SAX 方式并不矛盾，可以使用 DOM 来创建 SAX 事件流，也可以使用 SAX 来构造 DOM 树。可以这样简单地理解，首先使用 SAX 方式读取文件，读取的同时可以访问数据；如果需要更方便地操作整个文档，则可以在使用 SAX 方式读取的同时保存数据，这样最终读取完毕时也就构造出了 DOM 树。事实上大多数能够创建 DOM 树的 XML 解析器就是采用 SAX 方式来完成的。

注意 所谓事件驱动，是指一种基于回调（callback）机制的程序运行方法。如果读者对 Java 的代理事件模型比较清楚的话，就会很容易理解这种机制。后面章节会介绍这种机制。

5.2.2 解析 XML 的 API

针对 XML 解析，Java 领域存在非常多的 API，下面介绍常见的几种。

1. W3C DOM

使用 DOM 方式可以方便地操作整个 XML 文档，W3C 发布了针对 DOM 方式的一组 Java 接口，其中规范了以 DOM 方式操作 XML 文档的方法。W3C DOM 是业界认可的 DOM 规范，并且已经内置于 JDK 中，编写 Java 程序时不需要引入任何库就可以使用这个 API。W3C DOM API 所在的包是 org.w3c.dom。

2. SAX

SAX 同作为 API 的 DOM 一样，也是一个访问 XML 文档的接口，但 SAX 不是 W3C 的推荐标准，SAX 是由 OASIS（Organization for the Advancement of Structured Information Standards，结构化信息标准促进组织）下 xml.org 的 XML-DEV 邮件列表的成员开发维护，是一种社区性质的讨论产物。虽然如此，在 XML 中对 SAX 的应用丝毫不比 DOM 少，在运行中的各方面表现都优于 DOM API，几乎所有的 XML 解析器都会支持它，但 SAX 用起来不像 DOM 那样直观。SAX API 所在的包是 org.xml.sax。

SAX 解析器是一个把具体操作留给编程人员而把解释工作留给自己的一个编程模型。它不像 DOM 那样需要把整个 XML 文档加载到内存，而是逐行解释然后通过事件通知给解析程序，由具体的程序使用这些事件通知，最后加以处理。

3. JAXP

Java 和 XML 有许多相似的特性，比如平台无关性、可扩展性和可重用性等。将两者结合起来可以开发出具有更低开销的信息共享和数据交换的应用程序。针对这一要求 SUN 公司推出了一套轻量级的包装器 API 即 JAXP（Java API for XML Parsing，Java 解析 XML 应用接口）。实际上 JAXP 就是 Java 操作 XML 的标准规范，是基于 W3C DOM 和 SAX 规范创建的，支持使用 SAX、XSLT 和 DOM 的 XML 处理，并为 SAX 和 DOM API 提供一致性。JAXP 未对 SAX 和 DOM API 做任何改变，而是通过添加一些便利方法的方式，使得 Java 程序设计

人员在解析 XML 文档时不需要考虑具体使用哪一个 XML 解析器，从而更容易使用 XML API。通过 JAXP，可以使用任何与其兼容的 XML 解析器。

JAXP 没有重新定义 DOM 和 SAX，而是提供一种机制可以通过即插即用接口在 Java 应用程序中访问解析器。如图 5-3 所示。演示了 JAXP 的工作方式。

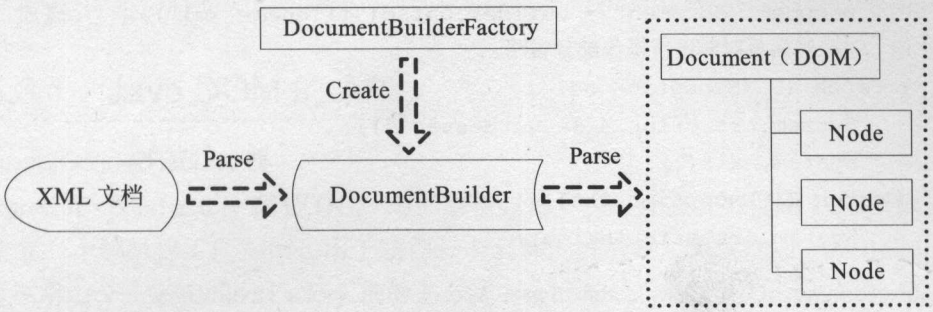


图 5-3　JAXP API

如图 5-3 所示，JAXP 使用 DocumentBuilderFactory 类创建文档构造器（DocumentBuilder），文档构造器接收输入的 XML 进行解析，以处理 DOM 文档。

JAXP 接口包含了以下三个包。

- org.w3c.dom：W3C 推荐的 DOM 规范接口。
- org.xml.sax：SAX 规范接口。
- javax.xml.parsers：解析器工厂工具，以供程序员获得并配置特殊的语法分析器。

上述三个包在 JDK 中自带，在 DOM 编程时不需要依赖其他包。

下述代码用于实现任务【描述 5.D.1】，通过实例代码，演示 JAXP 处理 XML 的一般过程。

【描述 5.D.1】 JAXPDemo.java

```
import java.io.*;
import javax.xml.parsers.*;
import org.w3c.dom.*;
import org.xml.sax.SAXException;

public class JAXPDemo {
    public static void main(String[] args) {
        // 得到 DOM 解析器的工厂实例
        DocumentBuilderFactory factory = DocumentBuilderFactory.newInstance();
        DocumentBuilder builder = null;
        try {
            // 从 DOM 工厂实例获得 DOM 解析器
            builder = factory.newDocumentBuilder();
        } catch (ParserConfigurationException pce) {
```

```
            System.err.println(pce);
            System.exit(1);
        }
        try {
            // 把要解析的 XML 文档读入到 DOM 解析器中
            Document document = builder.parse("filename.xml");
            // 对解析对象进行操作部分省略...
        } catch (SAXException se) {
            System.err.println(se.getMessage());
            System.exit(1);
        } catch (IOException ioe) {
            System.err.println(ioe);
            System.exit(1);
        }
    }
}
```

上述代码中并没有包含任何厂商专用的类库，而且，解析器虽然是 JAXP 创建的，但是抛出的异常是 SAXException，这是因为 JAXP 利用了很多 SAX 规范中的类。

4. JDOM

JDOM 是一个开源的 XML 解析类库，其提供了一种基于 Java 的特定文档对象模型。JDOM 并不符合 W3C DOM 或 JAXP 规范，它简化了与 XML 的交互，并且比使用 DOM 速度更快。

JDOM 与 DOM 主要有两方面不同：首先 JDOM 仅使用具体类而不使用接口，这在某些方面简化了 API，比 DOM 容易理解，但是也限制了灵活性；其次，JDOM 在 API 中大量使用了 Collections 类，简化了 Java 程序设计人员的工作。但是其自身不包含解析器。JDOM 通常使用 SAX2 解析器来解析和验证 XML 文档。JDOM API 所在的包是 org.jdom。

5. DOM4J

DOM4J 是一个开源的 XML 解析类库，提供了一种基于 Java 的特定文档对象模型，并且也提供对 W3C DOM、SAX 和 JAXP 的支持。DOM4J 还提供了许多超出基本 XML 文档表示的功能，包括集成的 XPath 支持、XSLT 支持、Xml Schema 支持，以及用于大文档或流化文档的基于事件的处理。DOM4J 通过大量使用接口和抽象类，并集中引入了 Collections 类，使得操作的灵活性非常高。DOM4JAPI 所在的包是 org.dom4j。

DOM4J 是一个非常优秀的 Java XML API，具有性能优异、功能强大和易用的特点，如 Hibernate 等许多著名的项目都是使用 DOM4J 来完成 XML 解析的。

注意 W3C DOM、SAX、JAXP 等都是接口规范，并没有实现 XML 的解析器，但是最新版的 JDK 内置了 Apache 的 Xerces 解析器，JAXP 默认会使用这个解析器。

5.3　DOM 编程

通俗地说，DOM 编程就是通过解析 XML 文档，为 XML 文档在逻辑上建立一个树模型，树的结点是一个个对象，通过存取这些对象就能够对 XML 文档的内容进行操作，维护 XML 文档中的数据。

5.3.1　Java DOM 的 API

1. DocumentBuilderFactory 类

DocumentBuilderFactory 类是 DOM 中的解析器工厂类，开发者要使用 DOM 操作 XML，首先要建立一个解析器工厂实例，用于获得一个具体的解析器对象（DocumentBuilder）。

通过调用 DocumentBuilderFactory 的静态方法 newInstance()来创建 DocumentBuilderFactory 类的新实例，代码如下：

```
DocumentBuilderFactory dbf = DocumentBuilderFactory.newInstance();
```

newInstance()方法为静态方法，如果该方法不能实例化，则抛出 FactoryConfigurationError 异常。

> 注意　DocumentBuilderFactory 类是非线程安全的，虽然应用程序可以使用 DocumentBuilderFactory 类的同一实例创建 DocumentBuilder 的实例，但开发者需确保 DocumentBuilderFactory 类的同一实例不被多个线程同时使用。

2. DocumentBuilder 类

DocumentBuilder 类是 DOM 中的解析器类，开发者使用该类可以获取 XML 文档的 DOM 对象实例。当获得一个解析器工厂类实例后，使用其静态方法 newDocumentBuilder()将可以获得一个 DOM 解析器对象，代码如下：

```
DocumentBuilder db = dbf.newDocumentBuilder();
```

这里，dbf 是解析器工厂类实例。上述代码如果不能创建 DocumentBuilder，那么就会抛出 ParseConfigurationException 异常。

> 注意　DocumentBuilder 类也是非线程安全的，开发者应确保该类的同一实例不被多个线程同时使用。

3. Document

Document（文档）接口代表整个文档，是对文档中的数据进行访问和操作的入口。文档中的所有 Node 对象都以一定的顺序包含在 Document 对象之内，排列成一个树形的结构，对 XML 文档的所有操作都通过对 Document 进行操作来完成。

解析器类 DocumentBuilder 的 parse()方法接受一个 XML 文档名作为输入参数，然后返回

一个 Document 实例，示例代码如下：

```
Document doc = db.parse("student.xml");
```

此外还可以把要解析的 XML 文档转化为输入流，然后将输入流对象作为 parse()方法的输入参数，以便 DOM 解析器解析它，示例代码如下：

```
InputStream is = new FileInputStream("message.xml");
Document doc = db.parse(is);
```

得到 Document 对象后，就可以通过 DOM 完成对 XML 文档的一系列操作。例如，通过代码得到的 XML 文档中所有<student>标签对应的 Node 对象。

```
NodeList nl = doc.getElementsByTagName("student");
```

上述代码通过使用 Document 对象的 getElementsByTagName()方法，返回一个 NodeList 对象，而 NodeList 列表中的每个元素都是一个 Node 对象，对应着 XML 文档中的一个标签元素即<student>标签。

除了查找结点工作之外，Document 还包含其他一系列方法，常用的方法及功能如表 5-1 所示。

表 5-1 Document 的部分方法

方法名	功能说明
createAttribute(String name)	用给定的属性名创建一个 Attr 对象，并可通过调用 setAttributeNode()方法来将其附加在某一个 Element 对象上
createElement(String name)	用给定的标签名创建一个 Element 对象，代表 XML 文档中的一个标签，然后就可以在这个 Element 对象上添加属性或进行其他的操作
createTextNode(String data)	用给定的字符串创建一个 Text 对象，Text 对象代表了标签或者属性中所包含的纯文本字符串。如果在一个标签内没有其他的标签，那么标签内的文本所代表的 Text 对象是这个 Element 对象的唯一子对象
getElementsByTagName(String name)	返回一个 NodeList 对象，它包含了所有给定标签名字的标签
getDocumentElement()	返回一个代表这个 DOM 树的根结点的 Element 对象，也就是代表 XML 文档根元素的那个对象

4. Node

Node（结点）接口代表文档树中的一个结点，是组成 XML 文档的最基本环节，因此在该接口中提供了大量的方法用于对 Node 对象实现各种访问和操作。其常用的方法及功能见表 5-2。

表 5-2 Node 的部分方法

方法名	功能说明
appendChild(Node node)	为这个结点添加一个子结点，并放在所有子结点的最后，如果这个子结点已经存在，则先把它删掉再添加进去

（续表）

方法名	功能说明
getFirstChild()	如果结点存在子结点，则返回第一个子结点
getLastChild()	如果结点存在子结点，则返回最后一个子结点
getNextSibling()	返回在 DOM 树中这个结点的下一个兄弟结点
getNodeName()	根据结点的类型返回结点的名称
getNodeType()	返回结点的类型
getNodeValue()	返回结点的值
hasChildNodes()	判断是否存在子结点
hasAttributes()	判断当前结点是否存在属性
getOwnerDocument()	返回当前结点所属的 Document 对象
insertBefore(Node new, Node ref)	在给定的一个子对象前再插入一个子对象
removeChild(Node node)	删除给定的子结点对象
replaceChild(Node new, Node old)	用一个新的 Node 对象代替给定的子结点对象

5. NodeList

NodeList（结点列表）接口是有序结点集的一种抽象，它是动态的，会随着文档的改变而自动改变。结点列表接口非常简单，其方法及功能见表 5-3。

表 5–3　NodeList 的方法

方法名	功能说明
getLength()	返回列表的长度。
item(int index)	返回指定位置的 Node 对象。

6. Element

Element 接口代表 XML 文档中的标签元素，其继承于 Node。由于标签元素可以包含属性，因此 Element 对象提供了对属性的存取操作。Element 的常用方法及功能见表 5-4。

表 5–4　Element 的部分方法

方法名	功能说明
getElementsByTagName(String name)	返回一个 NodeList 对象，它包含了该标签的子孙结点中具有给定标签名字的标签
getAttribute(String name)	返回标签中给定属性名的属性的值
getAttributeNode(String)	返回一个给定属性名称的 Attr 对象
hasAttribute(String name)	判断该标签中是否存在指定名称的属性
removeAttribute(String name)	删除该标签中指定的属性
setAttribute(String name, String value)	为标签增加指定名称和属性值的属性
setAttributeNode(Attr newAttr)	为标签增加指定属性，属性信息来源于指定的 Attr 对象

7. Attr

Attr 接口代表标签中的属性，其继承于 Node。在操作 Attr 对象时，需要注意的是，Attr 其实是被看做为包含它的 Element 对象的一部分，而并非作为 DOM 树中单独的一个结点出现，所以当使用从 Node 接口继承来的 getParentNode()，getPreviousSibling()和 getNextSibling()等方法时返回的值都将是 null。

5.3.2 DOM 应用

对于 XML 文件的常规操作，通过 DOM 提供的类（接口）及方法可方便地实现。

1. 遍历 XML

要遍历 XML，首先要解析 XML 文件并创建 Document 对象，然后通过调用 Node 接口提供的方法和属性即可获取各结点对象及子结点的相关值。

下述代码用于实现任务【描述 5.D.2】，通过载入给定 XML 文档，遍历并显示 XML 文档中的全部元素。

给定 XML 文档内容如下：

【描述 5.D.2】 student1.xml

```
<?xml version="1.0" encoding="gb2312"?>
<student-info>
    <student id="95001">
        <name>王明</name>
        <age>24</age>
        <address>青岛</address>
    </student>
    <student id="95002">
        <name>李刚</name>
        <age>24</age>
        <address>上海</address>
    </student>
</student-info>
```

对于上述 XML 文档，创建遍历代码如下。

【描述 5.D.2】 TraverseXML.java

```
import javax.xml.parsers.DocumentBuilder;
import javax.xml.parsers.DocumentBuilderFactory;
import org.w3c.dom.*;
public class TraverseXML {
    public static void main(String args[]) {
        try {
```

```
            // 得到 DOM 解析器的工厂实例
            DocumentBuilderFactory factory = DocumentBuilderFactory
                    .newInstance();
            // 从 DOM 工厂实例获得 DOM 解析器
            DocumentBuilder builder = factory.newDocumentBuilder();
            // 把要解析的 XML 文档读入到 DOM 解析器中
            Document doc = builder.parse("student1.xml");
            // 得到文档中名称为 PERSON 的元素的结点列表
            NodeList nl = doc.getElementsByTagName("student");
            // 遍历该列表，显示列表中元素及其子元素的名字
            for (int i = 0; i < nl.getLength(); i++) {
                Element node = (Element) nl.item(i);
                String value = null;
                // 获取元素的 id 属性
                System.out.println("id:" + node.getAttribute("id"));
                // 获取名称为 name 的元素列表中第一个元素的第一个子元素的值
                value = node.getElementsByTagName("name").item(0)
                        .getFirstChild().getNodeValue();
                System.out.println("name: " + value);
                value = node.getElementsByTagName("age").item(0)
                        .getFirstChild().getNodeValue();
                System.out.println("age: " + value);
                value = node.getElementsByTagName("address").item(0)
                        .getFirstChild().getNodeValue();
                System.out.println("address: " + value);
                System.out.println();
            }
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

执行结果如下：

```
id:95001
name: 王明
age: 24
address: 青岛

id:95002
name: 李刚
age: 24
```

```
address: 上海
```

上述代码中，采用了硬编码形式进行数据提取，对 XML 文档的结构和内容（元素的名称、元素的顺序、元素的个数等）有极大的依赖性，处理不够灵活。实际应用中要求 XML 处理程序能够应对各种符合规范的 XML 文档，此时可通过引入特定的算法来充分发挥 DOM 相关接口的作用。

下述代码用于实现任务【描述 5.D.3】，创建 Java 程序，对于任意给定的 XML 文档，实现 XML 文档的遍历。

【描述 5.D.3】 DOMPrinter.java

```
import java.io.File;
import java.io.IOException;
import javax.xml.parsers.DocumentBuilder;
import javax.xml.parsers.DocumentBuilderFactory;
import javax.xml.parsers.ParserConfigurationException;
import org.w3c.dom.*;
import org.xml.sax.SAXException;
public class DOMPrinter {
    /**
     * 采用递归调用，输出指定结点下的所有后代结点。 注：为了简单起见，
     * 只对处理文档结点、指令结点、元素结点、属性结点和文本结点进行了处理。
     **/
    public static void printNode(Node node) {
        short nodeType = node.getNodeType();
        switch (nodeType) {
        // 处理指令结点
        case Node.PROCESSING_INSTRUCTION_NODE:
            System.out.println("指令结点:<" + node.getNodeName() + " "
                    + node.getNodeValue()+">");
            break;
        // 处理元素结点
        case Node.ELEMENT_NODE:
            System.out.print("元素结点:<" + node.getNodeName());
            // 处理属性结点
            NamedNodeMap attrs = node.getAttributes();
            int attrNum = attrs.getLength();
            for (int i = 0; i < attrNum; i++) {
                Node attr = attrs.item(i);
                System.out.print( " "+attr.getNodeName() + "="
                        + attr.getNodeValue());
            }
            System.out.println(">");
```

```
            break;
        // 处理文本结点
        case Node.TEXT_NODE:
            if (node.getNodeValue()!=null
                && node.getNodeValue().trim().length()!=0) {
                System.out.println("文本结点:"+node.getNodeValue());
            }
            break;
        // 处理文档结点
        case Node.DOCUMENT_NODE:
            System.out.println("文档结点");
            break;
        default:
            System.out.println("未知结点");
            break;
        }
        // 递归调用，遍历复合的 node
        // for(Node child = node.getFirstChild();child != null;
        // child=child.getNextSibling()){
        // printNode(child);
        // }
        Node child = node.getFirstChild();
        while (child != null) {
            printNode(child);
            child = child.getNextSibling();
        }
        // 打印元素结束符
        if(node.getNodeType() == Node.ELEMENT_NODE ){
            System.out.println("结点结束:"+"</"+node.getNodeName()+">");
        }
    }
    public static void main(String[] args) {
        try {
            // 得到 DOM 解析器的工厂实例
            DocumentBuilderFactory dbf = DocumentBuilderFactory.newInstance();
            // 从 DOM 工厂实例获得 DOM 解析器
            DocumentBuilder db = dbf.newDocumentBuilder();
            Document doc = db.parse(new File("student1.xml"));
            // 调用递归方法
            printNode(doc);
        } catch (ParserConfigurationException e) {
            e.printStackTrace();
        } catch (SAXException e) {
```

```
            e.printStackTrace();
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

上述代码中，printNode()方法使用深度优先的方式来遍历 DOM 树。在深度优先的遍历方式中，当遇到一个结点时，处理序列沿着该结点下的路径一致延伸到叶子结点，然后回退，接着继续处理该结点的下一个兄弟结点。图 5-4 描述了上述处理过程。

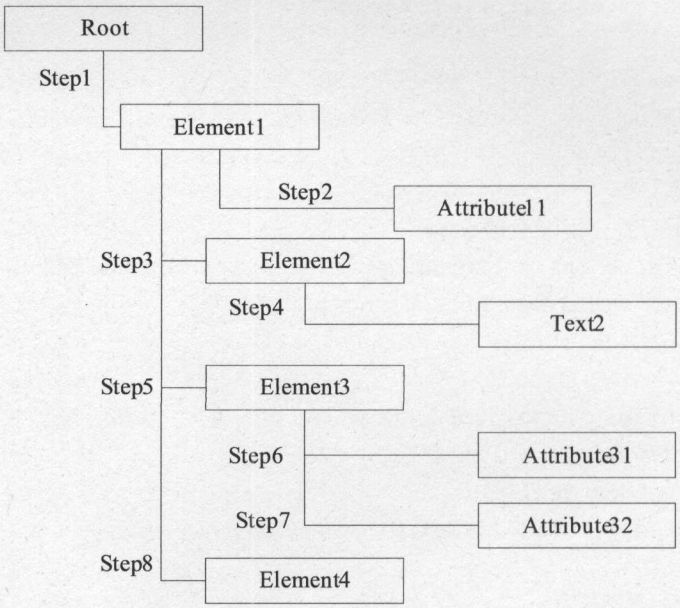


图 5-4　遍历 DOM 树

根元素（Root）是获取的第一个元素。然后转到 Element1，并检索它的属性（Attribute11）；接着提取 Element2，然后是它的文本值 Text2；序列中的下一个元素是 Element3，获取它的属性 Attribute31 和 Attribute32；最后程序转到 Element4。

另外，在 DOMPrinter.java 示例中，通过使用 Node 接口的符号常量来判断结点类型，表 5-5 列出了 Node 接口定义的 12 种结点类型及其作用。

表 5–5　Node 的部分符号常量

常量名	功能说明
ELEMENT_NODE	元素结点
ATTRIBUTE_NODE	属性结点
TEXT_NODE	文本结点
CDATA_SECTION_NODE	字符数据结点
ENTITY_REFERENCE_NODE	实体引用结点

（续表）

常量名	功能说明
ENTITY_NODE	实体结点
PROCESSING_INSTRUCTION_NODE	处理指令结点
COMMENT_NODE	注释结点
DOCUMENT_NODE	文档结点（根结点）
DOCUMENT_TYPE_NODE	文档类型结点
DOCUMENT_FRAGMENT_NODE	文档片段结点
NOTATION_NODE	记号结点

对于 student1.xml，上述代码执行结果如下：

```
文档结点
元素结点:<student-info>
元素结点:<student id=95001>
元素结点:<name>
文本结点:王明
结点结束:</name>
元素结点:<age>
文本结点:24
结点结束:</age>
元素结点:<address>
文本结点:青岛
结点结束:</address>
结点结束:</student>
元素结点:<student id=95002>
元素结点:<name>
文本结点:李刚
结点结束:</name>
元素结点:<age>
文本结点:24
结点结束:</age>
元素结点:<address>
文本结点:上海
结点结束:</address>
结点结束:</student>
结点结束:</student-info>
```

2. 创建 XML

通过 DOM 树，按照 XML 元素的层次结构，可以方便地创建 XML 文档。

下述代码用于实现任务【描述 5.D.4】，创建一个 XML 文档的 DOM 解析树，并将其保存成 XML 文档。

【描述 5.D.4】 CreateXML.java

```
import javax.xml.parsers.*;
import javax.xml.transform.*;
import org.w3c.dom.*;
import java.io.*;

public class CreateXML {
    public static void main(String args[]) {
        Document doc;
        // 声明 XML 文档中的各个元素对象
        Element students, student;
        Element name = null;
        Element address = null;
        Element tel = null;
        Element email = null;
        try {
            // 得到 DOM 解析器的工厂实例
            DocumentBuilderFactory dbf = DocumentBuilderFactory.newInstance();
            // 从 DOM 工厂实例获得 DOM 解析器
            DocumentBuilder builder = dbf.newDocumentBuilder();
            // 创建文档树模型对象
            doc = builder.newDocument();
            // 如果创建的文档树模型不为空
            if (doc != null) {
                // 创建 students 元素
                students = doc.createElement("students");
                // 创建 student 元素
                student = doc.createElement("student");
                // 设置 student 元素的属性 id 的值为 95003
                student.setAttribute("id", "95003");
                // 将 student 元素添加为 students 的子元素
                students.appendChild(student);
                // 创建 name 元素
                name = doc.createElement("name");
                // 将一个文本结点添加为 name 元素的子结点
                name.appendChild(doc.createTextNode("Mr. Zhang"));
                // 将 name 元素添加为 student 的子元素
                student.appendChild(name);
                address = doc.createElement("address");
                address.appendChild(doc.createTextNode("Shandong Qingdao"));
                student.appendChild(address);
                tel = doc.createElement("tel");
```

```
            tel.appendChild(doc.createTextNode("(0532)88888888"));
            student.appendChild(tel);
            email = doc.createElement("email");
            email.appendChild(doc.createTextNode("zhangfei@163.com"));
            student.appendChild(email);
            // 将 students 元素作为根元素添加到 XML 文档树中
            doc.appendChild(students);
            // 将内存中的文档树保存为 students.xml 文档
            Transformer transformer = TransformerFactory.newInstance()
                .newTransformer(); // 得到转换器
            // 设置换行
            transformer.setOutputProperty(OutputKeys.INDENT, "yes");
            // 写入文件
            transformer.transform(new DOMSource(doc),
                        new StreamResult(new File("d:/student.xml")));
            }
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

执行上述代码，会在项目目录下生成 students.xml 文档，其内容如下：

```
<?xml version="1.0" encoding="UTF-8"?>
<students>
<student id="95003">
<name>Mr. Zhang</name>
<address>Shandong Qingdao</address>
<tel>(0532)88888888</tel>
<email>zhangfei@163.com</email>
</student>
</students>
```

3. 修改 XML

通过 DOM 树可以方便地对 XML 文档进行常规的操作，如添加结点、修改结点值或删除结点。

下述代码用于实现任务【描述 5.D.5】，基于任务【描述 5.D.4】创建的 XML 文档，实现向该 XML 文档插入学生信息及其子元素。

【描述 5.D.5】 InsertXML.java

```
import java.io.*;
import javax.xml.parsers.*;
```

```
import javax.xml.transform.*;
import org.w3c.dom.*;
public class InsertXML {
    public static void main(String args[]) {
        Element students = null;
        Element student = null;
        Element name = null;
        Element address = null;
        Element tel = null;
        Element email = null;
        try {
            // 得到 DOM 解析器的工厂实例
            DocumentBuilderFactory factory = DocumentBuilderFactory
                    .newInstance();
            // 从 DOM 工厂实例获得 DOM 解析器
            DocumentBuilder builder = factory.newDocumentBuilder();
            // 把要解析的 XML 文档读入到 DOM 解析器中
            Document doc = builder.parse("students.xml");
            // 得到文档中名称为 students 的元素的结点列表
            NodeList nl = doc.getElementsByTagName("students");
            students = (Element) nl.item(0);
            // 创建名称为 student 的元素
            student = doc.createElement("student");
            // 设置元素的 id 属性值为 95005
            student.setAttribute("id", "95005");
            // 创建名称为 NAME 的元素
            name = doc.createElement("name");
            // 创建名称为 Tony Blair 的文本结点并将其作为子结点添加到 name 元素中
            name.appendChild(doc.createTextNode("Tony Blair"));
            // 将 name 元素作为子元素添加到 person 元素中
            student.appendChild(name);
            address = doc.createElement("address");
            address.appendChild(doc.createTextNode("Yantai"));
            student.appendChild(address);
            tel = doc.createElement("tel");
            tel.appendChild(doc.createTextNode("(0533)6666666"));
            student.appendChild(tel);
            email = doc.createElement("email");
            email.appendChild(doc.createTextNode("blair@163.com"));
            student.appendChild(email);
            // 将 student 元素作为子元素添加到 XML 文档树的根结点 students 中
            students.appendChild(student);
            // 将内存中的文档树通过文件流生成 students_new.xml 文档
```

```
                Transformer transformer = TransformerFactory.newInstance()
                    .newTransformer(); // 得到转换器
                // 设置换行
                transformer.setOutputProperty(OutputKeys.INDENT, "yes");
                // 写入文件
                transformer.transform(new DOMSource(doc),
                        new StreamResult(new File("students.xml")));
            } catch (Exception e) {
                e.printStackTrace();
            }
        }
}
```

执行上述代码，会在项目目录下生成 students_new.xml 文档，其内容如下：

```
<?xml version="1.0" encoding="UTF-8"?>
<students>
<student id="95003">
<name>Mr. Zhang</name>
<address>Shandong Qingdao</address>
<tel>(0532)88888888</tel>
<email>zhangfei@163.com</email>
</student>
<student id="95005">
<name>Tony Blair</name>
<address>Yantai</address>
<tel>(0533)6666666</tel>
<email>blair@163.com</email>
</student>
</students>
```

对于元素进行修改（删除）操作，只需获取要修改（删除）的结点，然后在此结点上按 XML 元素的层次关系调用相关的方法进行相应操作即可。

5.3.3　DOM 解析器属性

通过设置 DocumentBuilderFactory 实例的属性，可用于配置解析器，这些属性限制了解析器在遍历 XML 文件时需要关注或可忽略的地方。在配置解析器属性时需要特别注意的一点是，只有在请求文档构建器之前进行属性设置才有效，也就是说，如果文档构建器已经实例化，然后再去设置这些属性，文档构建器将不会自动更新。

以 dbf 作为文档构建器实例，介绍配置解析器时常用的属性、功能及设置方法。

- Coalesce

该属性指定解析器是否把字符数据（CDATA）转换成 Text 结点。当属性值为 true 时，解析器会把每个 CDATA 结点都转换成 Text 结点，并将其添加到相邻的文本结点。设置方法如下所示：

```
dbf.setCoalescing(true);
```

- ExpandEntityReferences

  该属性指定解析器是否展开实体引用结点。当属性值设置为 true 时，解析器将展开实体引用结点。该属性通过如下方法设置：

```
dbf.setExpandEntityReferences(true);
```

- IgnoreComments

  该属性指定解析器是否忽略 XML 文档中的注释。当属性值为 true 时，解析器将忽略文档中的注释。该属性通过如下方法设置：

```
dbf.setIgnoringComments(true);
```

- IgnoreElementContentWhitespace

  该属性指定解析器在处理 XML 文档时，是否忽略其中的无意义空格（文档空白）。当值为 true 时，解析器会删去 XML 中可忽略的空格（空白）。该属性通过如下方法设置：

```
dbf.setIgnoringElementContentWhitespace(true);
```

- NamespaceAware

该属性指定解析器是否需要理解命名空间。如果该属性设置为 true，由工厂创建的任何解析器将支持 XML 中的命名空间。该属性通过如下方法设置：

```
dbf.setNamespaceAware(true);
```

- Validate

  该属性指定解析器在解析 XML 文件时是否需要对其进行验证。该属性通过如下方法设置：

```
dbf.setValidating(true);
```

下述代码用于实现任务【描述 5.D.6】，创建验证程序，通过带 DOCTYPE 声明的 XML 文件检查解析器的验证情况。

给定 XML 文档内容如下所示。

【描述 5.D.6】 student.xml

```
<?xml version="1.0" encoding="GB2312" standalone="yes"?>
```

```
<!DOCTYPE student [
    <!ELEMENT student (name)>
    <!ELEMENT name (#PCDATA)>
]>
<student>
    <name>9</name>
</student>
```

创建验证的 DOM 解析器的代码如下所示。

【描述 5.D.6】 ValidateXML.java

```
import java.io.*;
import javax.xml.parsers.*;
import org.xml.sax.*;
import org.w3c.dom.*;
public class ValidateXML {
    ValidateXML(String filename) {
        // 得到 DOM 解析器的工厂实例
        DocumentBuilderFactory dbf = DocumentBuilderFactory.newInstance();
        // 设置 DOM 解析器的配置选项
        dbf.setValidating(true);
        dbf.setIgnoringComments(true);
        dbf.setIgnoringElementContentWhitespace(true);
        dbf.setNamespaceAware(true);
        dbf.setExpandEntityReferences(true);
        // 从 DOM 工厂实例获得 DOM 解析器
        DocumentBuilder builder = null;
        try {
            builder = dbf.newDocumentBuilder();
        } catch (ParserConfigurationException pce) {
            System.err.println(pce);
            System.exit(1);
        }
        // 设置错误处理器
        builder.setErrorHandler(new XMLErrorHandler());
        // 把要解析的 XML 文档读入到 DOM 解析器中
        Document parsedDocument = null;
        try {
            parsedDocument = builder.parse(new File(filename));
        } catch (SAXException se) {
            System.err.println(se.getMessage());
            System.exit(1);
        } catch (IOException ioe) {
```

```
            System.err.println(ioe);
            System.exit(1);
        }
        printNode(parsedDocument);
    }
    void printNode(Node node) {
        // 打印结点信息
    }
    // 错误处理类
    private static class XMLErrorHandler implements ErrorHandler {
        // 返回描述解析异常的字符串
        private String getParseExceptionInfo(SAXParseException spe) {
            String systemid = spe.getSystemId();
            if (systemid == null) {
                systemid = "null";
            }
            String info = "URI=" + systemid + " Line=" + spe.getLineNumber()
                    + ": " + spe.getMessage();
            return info;
        }
        public void warning(SAXParseException spe) throws SAXException {
            System.out.println("Warning: " + getParseExceptionInfo(spe));
        }
        public void error(SAXParseException spe) throws SAXException {
            String message = "Error: " + getParseExceptionInfo(spe);
            throw new SAXException(message);
        }
        public void fatalError(SAXParseException spe) throws SAXException {
            String message = "Fatal Error: " + getParseExceptionInfo(spe);
            throw new SAXException(message);
        }
    }
    public static void main(String args[]) {
        ValidateXML test = new ValidateXML("student.xml");
    }
}
```

由于给定的 XML 文档符合规范，所以执行上述代码，不会有错误信息输出。

如果调整 student.xml 文档的内容，使其违反 XML 规范，如下述代码所示。

```
<?xml version="1.0" encoding="GB2312" standalone="yes"?>
<!DOCTYPE student [
    <!ELEMENT student (name)>
```

```
    <!ELEMENT name (#PCDATA)>
]>
<student>
    <age>9</age>
</student>
```

再次执行验证程序，控制台会输出如下信息：

```
Error: URI=file:/E:/Workspaces8/ch05/student.xml Line=7: Element "student" does
not allow "age" here.
```

如果此时将验证程序的验证属性设置语句“dbf.setValidating(true)”注释掉，再次执行此程序，则不会有错误信息输出。

小结

通过本章的学习，学生应该能够学会：

- XML DOM 是 XML Document Object Model 的缩写。
- DOM 定义了访问和处理 XML 文档的标准方法。
- 在 DOM 中，将 XML 逻辑结构描述成树（DOM 树）。
- XML 中共有 12 种结点类型，其中最常见的结点类型有 5 种：文档、元素、属性、文本和注释。
- DOM 是文档驱动的，不适于处理大型 XML 文件。
- 主要有 DOM 和 SAX 两种解析 XML 文档的方式。
- JAXP 提供一种机制可以通过即插即用接口在 Java 应用程序中访问 DOM 解析器。
- JAXP 接口包含了三个包：org.w3c.dom、org.xml.sax、javax.xml.parsers。
- 常用的 DOM 接口有 DocumentBuilderFactory、DocumentBuilder、Document、Node、Element 等。
- DocumentBuilderFactory 类、DocumentBuilder 类是非线程安全的，开发者应确保该类同一实例不被多个线程同时使用。
- 通过设置 DocumentBuilderFactory 实例的属性，可用于配置解析器，用于控制对 XML 文件的解析粒度

练习

1. 下述关于 DOM 的描述，错误的是_____。

   A. 在 DOM 中，一般将 XML 逻辑结构描述成树

   B. DOM 提供的对象和方法可以和任何编程语言（Java、C++、VB）一起使用

C. DOM 是文档驱动的，不需将整个 XML 文件读入内存

D. SAX 是基于事件驱动的，适于处理大型 XML 文件

2. 下述关于 JAXP 的描述，错误的是_____。

A. JAXP（Java API for XML Parsing）是 Java 处理 XML 的基础类库

B. JAXP 是重新定义 DOM 和 SAX 后创建的一套新类库

C. JAXP 接口包含了三个包：org.w3c.dom、org.xml.sax、javax.xml.parsers

D. JAXP 是基于 W3C 规范创建的，JAXP 支持使用 SAX、XSLT 和 DOM 的 XML 处理

3. 基于 JAXP，下述关于 DOM 解析器属性的描述，正确的是_____。

A. Coalesce 属性指定解析器是否把字符数据（CDATA）转换成 Text 结点

B. ExpandEntityReferences 属性指定解析器是否展开实体引用结点

C. Validate 属性指定解析器在解析 XML 文件时是否需要对其进行验证

D. IgnoreComments 属性指定解析器是否忽略 XML 文档中的注释

4. XML DOM 是 XML Document Object Model 的缩写，即_____。

5. 在 DOM 模型中，_____指定 DOM 树的顶部结点，_____是 XML 的基本构件，_____是 DOM 树的重心，构成了 DOM 树的核心结构。

6. 在 JAXP 中，_____类是 DOM 中的解析器工厂类，_____类是 DOM 中的解析器类，_____接口代表整个文档，是对文档中的数据进行访问的和操作的入口，_____接口代表文档树中的一个结点，_____接口代表 XML 文档中的标签元素。

7. 对于下列 XML 文档：

```
<?xml version="1.0" encoding="GB2312"?>
<Orders>
    <Order orderID="A001" orderDate="2010-7-20">
        <name>玩具</name>
        <number>16</number>
        <city>上海</city>
        <zip>200000</zip>
        <phoneno>13577778888</phoneno>
    </Order>
    <Order orderID="A002" orderDate="2010-7-22">
        <name>文具</name>
        <number>17</number>
        <city>青岛</city>
        <zip>266000</zip>
        <phoneno>0532-66667777</phoneno>
    </Order>
    <Order orderID="A004" orderDate="2010-8-22">
        <name>衣物</name>
        <number>56</number>
```

```
        <city>青岛</city>
        <zip>266000</zip>
        <phoneno>0532-88889999</phoneno>
    </Order>
</Orders>
```

基于 DOM，创建一个 XML 处理程序，要求实现如下功能：

- 载入并遍历给定 XML 文档的内容。
- 能够根据用户输入的订单号，查找相应的元素，并显示其信息。
- 将内存中的最新数据写入到新文件中。
- 上述功能要求使用菜单驱动方式实现。

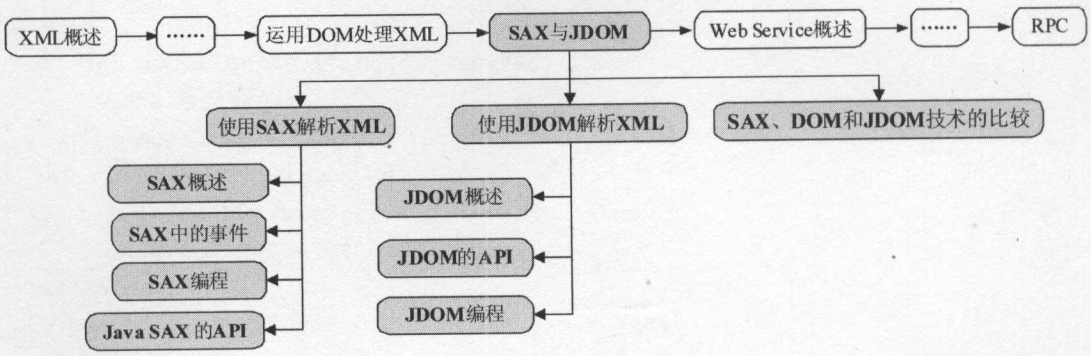
8. 请描述使用 DOM 处理 XML 文档的优越性。
9. 请描述常见的基于 JAVA 的 XML 解析器及其各自的特点（最少描述三个）。

第 6 章　SAX 与 JDOM

本章目标

- 了解 SAX 的概念
- 了解 SAX 的实现机制
- 掌握 SAX 的常用事件
- 掌握 SAX 的常用 API
- 掌握使用 SAX 处理 XML 的基本步骤
- 了解 JDOM 的概念和基本结构
- 掌握 JDOM 的常用 API
- 掌握使用 JDOM 处理 XML 的基本步骤

学习导航



任务描述

【描述 6.D.1】

创建 Java 程序，演示使用 SAX 处理 XML 的过程及 SAX 常用事件的触发次序。

【描述 6.D.2】

创建解析程序，统计给定 XML 文件中各个标签出现的次数。

【描述 6.D.3】

创建解析程序，演示 SAX 错误处理的应用。

【描述 6.D.4】

结合现有实例，构造过滤器，通过过滤掉指定 XML 文档中的元素来演示 XMLFilter 的作用。

【描述 6.D.5】

通过 JDOM 创建一个 XML 文档，并将其保存在本地磁盘 D。

【描述 6.D.6】

使用 JDOM 载入给定的 XML 文档，然后遍历并显示 XML 文档中的全部元素。

6.1 使用 SAX 解析 XML

SAX 同 DOM 一样，可以对 XML 文档进行解析。

6.1.1 SAX 的实现机制

SAX 是基于事件驱动的，类似于标签库的处理机制，在标签开始、标签结束以及错误发生等地方调用相应的接口实现方法。与 DOM 比较而言，SAX 是一种轻量型的方法。实现了 SAX API 的 XML 解析器根据解析到的 XML 文档的不同特征产生相关事件。通过在 Java 代码中捕捉这些事件，就可以写出由 XML 数据驱动的程序（类似于 AWT 中的事件驱动机制，通过事件驱动来识别 XML 文档的内容）。

由于 SAX API 是基于事件驱动的，SAX 是顺序、层次化地分析 XML 文档，着眼于当前的事件连续的处理，不是全部文档都读入内存，因此它并不需要读入整个文档。而文档的读入过程也就是 SAX 的解析过程，即读入文档的过程和解析的过程是同时进行的，这一点与 DOM 区别很大。

SAX 基于流式处理，在遇到一个标签的时候，它并不会记录下以前所碰到的标签，即在 startElement()方法中，所有能获知的信息就是标签的名字和属性，至于标签的嵌套结构、上层标签的名字、是否有子元属等其他与结构相关的信息，都是不得而知的，都需要通过程序来完成，这使得 SAX 在编程处理上没有 DOM 那么方便，而且编程复杂一些。但 SAX 的“即读即处理”，没有冗余的对象模型，对于大多数 XML 文档，不需要读到文档结尾即可得到相关信息，使得 SAX 处理速度和内存的占用要优于 DOM 模型。

注意 到现在为止，SAX 的版本已经发展到 2.0。在这个最新版本中增加了对命名空间的支持，而且可以通过对 features 以及 properties 的设置来对解析器进行全面的配置，这其中包括设置解析器是否对文档进行有效性验证，以及怎样来处理带有名称空间的元素名称等。SAX1 中的接口已经不再使用了，这里只讨论有关 SAX2 的开发。在本文中提到 SAX 只是指 SAX2。

6.1.2 SAX 中的事件

SAX 是对事件进行处理的，它把 XML 的解析当做事件触发来处理，采用的是事件驱动。SAX 的事件驱动模型如图 6-1 所示。

SAX 解析器中主要有 startDocument、startElement、characters、endElement、endDocument 5 种事件。各种事件及其处理方法如下。

1. startDocument 事件

该事件表明 SAX 解析器发现了文档的开始。该事件没有传递任何信息，只是表明解析器开始扫描文档。调用这个方法，可以在其中做一些预处理的工作。其声明的语法格式如下所示：

```
public void startDocument() throws SAXException
```

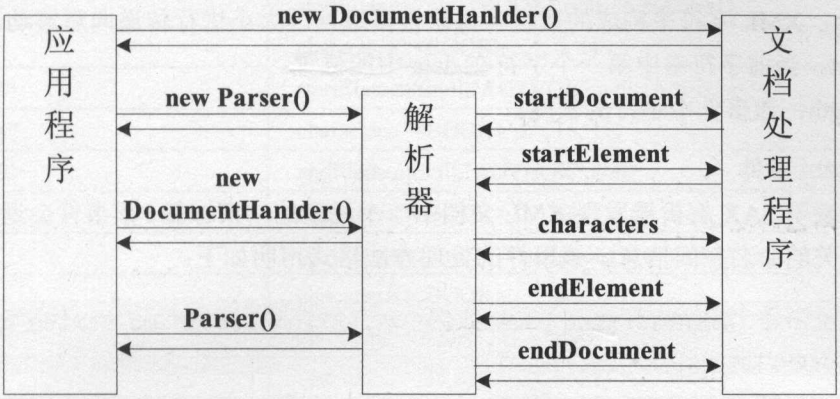


图 6-1　SAX 事件驱动模型

2. endDocument 事件

该事件表明 SAX 解析器发现 XML 文档的结尾。可使用这个方法做一些善后的工作。该事件的处理方法格式声明如下：

```
public void endDocument() throws SAXException
```

3. startElement 事件

该事件表明 SAX 解析器发现 XML 文件中一个元素的起始标签。该事件会返回该元素的名称、该元素属性的名称和值。如果应用程序需要查找 XML 中的某个元素的内容，该事件将会通知该元素何时开始。该事件的处理方法格式声明如下：

```
public void startElement(String namespaceURI,String localName,
String qName,Attributes atts) throws SAXException
```

该事件处理程序包含以下 4 个参数。

- namespaceURI：命名空间 URI。如果 XML 文档没有使用命名空间，该参数为空（null）。
- localName：元素名前缀（如果支持命名空间），如<util:list>将返回 util。
- qName：元素的限定名，<bean>返回 bean，而<util:list>返回 list（如果支持命名空间）。
- atts：包含该元素所有属性列表的对象。该对象提供了几种方法获取属性的名称和值，以及给元素的属性个数。

4. characters 事件

该事件表明 SAX 解析器发现 XML 文档中一个元素的文本信息。返回的信息包括一个字符串数组、该数组的偏移量和一个长度变量，通过这 3 个变量就可以访问所需处理的文本信息。其处理方法格式声明如下：

```
public void characters(char[] data,int start,int length) throws SAXException
```

该事件处理程序包含如下 3 个参数。

- data：XML 字符串构成的数组（每次根据缓冲区大小进行传递向后滚动）。
- start：当前字符串中第一个字符在 data 中的位置。
- length：该事件中字符的长度。

5. endElement 事件

该事件表明 SAX 解析器发现 XML 文档中一个元素的结束标签。该事件会返回该元素的名称以及相关的命名空间信息。该事件的处理方法格式声明如下：

```
public void endElement(String namespaceURI,String localName,String qName)
throws SAXException
```

该事件处理程序包含以下 3 个参数。

- namespaceURI：命名空间 URI。如果 XML 文档没有使用命名空间，该参数为 null。
- localName：元素名前缀（如果支持命名空间）。
- qName：元素的限定名。

6. 解析演示

下面以如下 XML 文档为例，讲解 SAX 的解析过程。

```
<BOOK>
    <AUTHOR>White Jeny</AUTHOR>
    <TITLE>Cloud</TITLE>
    <LINE>Adam</LINE>
</BOOK>
```

使用 SAX 解析之前，需要向 XMLReader 注册一个 ContentHandler，也就是相当于一个事件监听器，在 ContentHandler 中定义了很多方法，比如 startDocument()，它定制了当在解析过程中，遇到文档开始时应该处理的事件。当 XMLReader 读到合适的内容，就会抛出相应的事件，并把这个事件的处理权代理给 ContentHandler，调用其相应的方法进行响应。

当 XMLReader 读到<BOOK>标签时，就会调用 ContentHandler.startElement()方法，并把标签名 BOOK 作为参数传递过去，并在实现的 startElement()方法中需要做相应的动作，以处理当<BOOK>出现时应该做的事情。

各个事件随着解析的过程（也就是文档读入的过程），一个个顺序地被抛出，相应的方法也会被顺序地调用，最后，当解析完成，方法都被调用后，对文档的处理也就完成了。

如表 6-1 所示，列出了在解析上述 XML 文档时依次被调用的方法。

表 6-1　SAX 解析过程

元素名	功能说明
{文档开始}	startDocument()

（续表）

元素名	功能说明
<BOOK>	startElement(null,"BOOK",null,{Attributes})
" "	characters("<BOOK> ...", 6, 1)
<AUTHOR>	startElement(null,"AUTHOR",null,{Attributes})
"White Jeny"	characters("<BOOK> ...", 15, 10)
</AUTHOR>	endElement(null,"AUTHOR",null)
" "	characters("<BOOK> ...", 34, 1)
<TITLE>	startElement(null,"TITLE",null,{Attributes})
"Cloud"	characters("<BOOK> ...", 42, 5)
</TITLE>	endElement(null,"TITLE",null)
" "	characters("<BOOK> ...", 55, 1)
<LINE>	startElement(null,"LINE",null,{Attributes})
"Adam"	characters("<BOOK> ...", 62, 4)
</LINE>	endElement(null,"LINE",null)
" "	characters("<BOOK> ...", 67, 1)
</BOOK>	endElement(null,"BOOK",null)
{文档结束}	endDocument()

6.1.3　Java SAX 的 API

SAX 的 API 中最重要的组成部分是接口 XMLReader 和类 DefaultHandler。

1. SAXParserFactory

SAXParserFactory 类是 SAX 解析器工厂类，开发者要使用 SAX 操作 XML，首先要建立一个解析器工厂实例，以便利用这个工厂类实例来获得一个具体的解析器对象（SAXParser）。其功能与 DOM 中的 DocumentBuilderFactory 类相同。

通过调用 SAXParserFactory 的静态方法 newInstance()来创建 SAXParserFactory 类的新实例，代码如下：

```
SAXParserFactory spf = SAXParserFactory.newInstance();
```

newInstance()方法为静态方法，如果该方法不能实例化，则抛出 FactoryConfigurationError 异常。

2. SAXParser

SAXParser 是解析器类，其内部封装 XMLReader（包装器），并通过定义多种形式的 parse()方法解析来自文件、输入流、URL 中的 XML，并把解析事件报告给特定的处理器（DefaultHandler）。当获得一个解析器工厂类实例后，使用它的静态方法 newSAXParser()将可以获得一个 SAXParser 对象，代码如下：

```
SAXParser sp = spf.newSAXParser();
```

这里，spf 是解析器工厂类实例。上述代码如果不能创建 SAXParser，那么就会抛出 ParseConfigurationException 异常。

3. DefaultHandler

类 DefaultHandler 实现了四个核心 SAX2 处理程器接口（ContentHandler、ErrorHandler、DTDHandler 和 EntityResolver）中的所有回调，该类需要开发者自行继承，并实现“感兴趣”的方法。

4. ContentHandler

内容处理接口，当遇到 XML 文档中的标签时，就将会调用这个接口中的 startDocument()、endDocument()、startElement()和 endElement()方法。当遇到 XML 文档中的元素内容时，将调用 characters()方法。

5. ErrorHandler

错误处理接口，当遇到不同类型错误的时候分别调用相应的错误处理方法，这些方法包括：error()、fatalError()和 warning()。

6. DTDHandler

DTD 处理接口，该接口所定义的方法只用于处理 DTD 信息。

7. EntityResolver

该接口中的 resolveEntity 方法只在遇到 URI 标识数据的时候才被调用。

图 6-2 描述了 SAX API 的结构图。

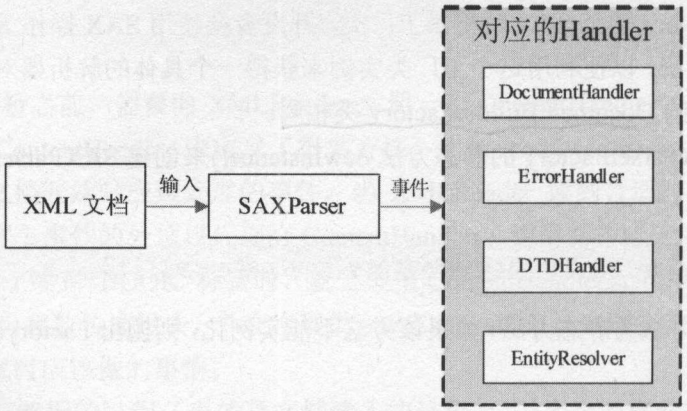


图 6-2　SAX API 的结构图

6.1.4　SAX 编程

使用 SAX 处理 XML 的主要任务是创建一个实现 ContentHandler 接口的类，该类用以将分析 XML 文档时所发生的 SAX 事件分发给处理程序的回调接口。为了方便，也可以继承

DefaultHandler 适配器类，该类实现了 ContentHandler 接口的方法。

使用 SAX 处理 XML 的一般步骤如下。

01 实现 ContentHandler 接口并填写“感兴趣”的事件代码（也可以继承 DefaultHandler 类，该类实现了 ContentHandler 接口）。

02 获得 SAXParserFactory 实例。

03 由 SAXParserFactory 获得 SAXParser 实例。

04 调用 SAXParser 实例的 parser()方法，此时需给出“XML 源”和一个 DefaultHandler 实现类的实例。

在 SAX API 中有两个主要包，即 org.xml.sax 和 org.xml.sax.helper。其中 org.xml.sax 中主要定义了 SAX 的一些基础接口，而在 org.xml.sax.helper 中则是一些方便开发人员使用的帮助类。

1. SAX 处理演示

下述代码用于实现任务【描述 6.D.1】，创建 Java 程序，演示使用 SAX 处理 XML 的过程及 SAX 常用事件的触发次序。

【描述 6.D.1】 SAXExample.java

```
import org.xml.sax.*;
import org.xml.sax.helpers.*;
import java.io.*;
import javax.xml.parsers.SAXParser;
import javax.xml.parsers.SAXParserFactory;
//重载 DefaultHandler 类的方法以拦截 SAX 事件通知。
public class SAXExample extends DefaultHandler {
    // 存储文本内容
    private StringBuffer buf;
    public void startDocument() throws SAXException {
        buf = new StringBuffer();
        System.out.println("*******开始解析文档*******");
    }
    public void endDocument() throws SAXException {
        System.out.println("*******解析文档结束*******");
    }
    public void startElement(String namespaceURI, String localName,
            String fullName, Attributes attributes) throws SAXException {
        System.out.println("元素: " + "[" + fullName + "]" + " 开始解析!");
        // 打印出属性信息
        for (int i = 0; i < attributes.getLength(); i++) {
            System.out.println("\t 属性名称:" + attributes.getLocalName(i)
```

```
            + " 属性值:"+ attributes.getValue(i));
        }
    }
    public void endElement(String namespaceURI, String localName,
            String fullName) throws SAXException {
        // 打印出非空的元素内容并将 StringBuffer 清空
        String nullStr = "";
        if (!buf.toString().trim().equals(nullStr)) {
            System.out.println("\t 内容是: " + buf.toString().trim());
        }
        buf.setLength(0);
        // 打印元素解析结束信息
        System.out.println("元素: " + "[" + fullName + "]" + " 解析结束!");
    }
    public void characters(char[] chars, int start, int length)
            throws SAXException {
        // 将元素内容累加到 StringBuffer 中
        buf.append(chars, start, length);
    }
    public static void main(String[] args) {
        String xmlFilename = "SAXExample.xml";
        // 创建 SAX 解释器工厂对象
        SAXParserFactory spf = SAXParserFactory.newInstance();
        SAXParser saxParser = null;
        try {
            // 通过工厂对象创建 SAX 解释器 SAXParser 对象
            saxParser = spf.newSAXParser();
        } catch (Exception ex) {
            System.err.println(ex);
        }
        try {
            // 使用 SAX 解释器加载 XML 文档,把实现了 DefaultHander 的实例对象装入解释器中
            saxParser.parse(new File(xmlFilename), new SAXExample());
        } catch (SAXException se) {
            System.err.println(se.getMessage());
        } catch (IOException ioe) {
            System.err.println(ioe);
        }
    }
}
```

上述代码是使用 SAX 解析一个 XML 文档的基本过程，由于 DefaultHandler 实现了四个核心 SAX2 处理程器接口，继承 DefaultHandler 的好处是不必实现所有的方法，该程序只重

写了解析过程中的基本事件。该实例很简单，但是足以阐述 SAX 编程的基本思路。给定如下 XML 文档：

【描述 6.D.1】 SAXExample.xml

```
<?xml version="1.0" encoding="gb2312"?>
<student-info>
    <student id="95001">
        <name>王明</name>
        <age>24</age>
        <address>青岛</address>
    </student>
</student-info>
```

上述程序的执行结果如下：

```
*******开始解析文档*******
元素: [student-info] 开始解析!
元素: [student] 开始解析!
    属性名称:id 属性值:95001
元素: [name] 开始解析!
    内容是: 王明
元素: [name] 解析结束!
元素: [age] 开始解析!
    内容是: 24
元素: [age] 解析结束!
元素: [address] 开始解析!
    内容是: 青岛
元素: [address] 解析结束!
元素: [student] 解析结束!
元素: [student-info] 解析结束!
*******解析文档结束*******
```

2. 统计文档标签个数

下述代码用于实现任务【描述 6.D.2】，创建解析程序，统计给定 XML 文件中各个标签出现的次数。

【描述 6.D.2】 SAXCounter.java

```
import org.xml.sax.helpers.DefaultHandler;
import javax.xml.parsers.*;
import org.xml.sax.*;
import java.util.*;
import java.io.*;
//实现 ContentHandler 接口并填写事件代码
```

```
public class SAXCounter extends DefaultHandler {
    // 这个 HashMap 用来记录 tag 出现的次数
    private HashMap<String, Integer> tags;
    public static void main(String[] args) {
        String xmlFilename = "SAXCounter.xml";
        // 创建 SAX 解释器工厂对象
        SAXParserFactory spf = SAXParserFactory.newInstance();
        SAXParser saxParser = null;
        try {
            // 通过工厂对象创建 SAX 解释器 SAXParser 对象
            saxParser = spf.newSAXParser();
        } catch (Exception ex) {
            System.err.println(ex);
        }
        try {
            //使用 SAX 解释器加载 XML 文档,把实现了 DefaultHander 的实例对象装入到解释器中
            saxParser.parse(new File(xmlFilename), new SAXCounter());
        } catch (SAXException se) {
            System.err.println(se.getMessage());
        } catch (IOException ioe) {
            System.err.println(ioe);
        }
    }
    // 解释器会回调程序中的事件过程，本方法可以实现处理文档前的工作
    public void startDocument() throws SAXException {
        // 初始化 HashMap
        tags = new HashMap<String, Integer>();
    }
    // 对每一个开始元素进行处理
    public void startElement(String namespaceURI, String localName,
            String qName, Attributes atts) throws SAXException {
        // 获得标签的名称字符串
        String key = qName;
        // 识别在 HashMap 中是否有该同名的标签存在
        Integer value = tags.get(key);
        // 如果是新发现的标签，这在 HashMap 中添加一条新的标签记录;
        // 如果以前记录过，得到其计数值，并加 1
        if (value == null) {
            tags.put(key, new Integer(1));
        } else {
            value++;
            tags.put(key, value);
        }
```

```
    }
    // 解析完成后的统计工作
    public void endDocument() throws SAXException {
        Set<Map.Entry<String, Integer>> set = tags.entrySet();
        for (Map.Entry<String, Integer> entry : set) {
            System.out.println("Tag <" + entry.getKey() + "> occurs "
                    + entry.getValue() + " times");
        }
    }
}
```

上述程序的思路很清晰，在文档开始处（startDoucment()），初始化统计容器（HashMap），在解析 XML 期间，当发现标签元素就对容器中的数据进行相应调整，并在文档结束处（endDocument()）进行遍历输出。给定如下 XML 文档：

【描述 6.D.2】 SAXCounter.xml

```
<?xml version="1.0" encoding="gb2312"?>
<student-info>
    <student id="95001">
        <name>王明</name>
        <age>24</age>
        <address>青岛</address>
    </student>
    <student id="95002">
        <name>李刚</name>
        <age>24</age>
        <address>上海</address>
    </student>
</student-info>
```

上述程序的执行结果如下：

```
Tag <student> occurs 2 times
Tag <address> occurs 2 times
Tag <age> occurs 2 times
Tag <name> occurs 2 times
Tag <student-info> occurs 1 times
```

3. 错误处理

上述 SAX 应用只是实现了 SAX 处理的基本方法，如果在解析过程中出现了错误需要实现 ErrorHandler 接口来处理。通过注册错误处理程序，可以捕获相应错误信息并加以处理，以增强程序的健壮性。ErrorHandler 接口定义了 3 个方法，分别是 warning()、error()和 fatalError()，其作用如下所示。

- warring()：该方法用来报告在 XML1.0 规范中定义的非错误(error)或者致命错误(fatal error)的错误状态。对于该类错误默认是什么也不做，SAX 解析器调用该方法后可继续解析处理文档。
- error()：该方法对应于 XML 1.0 规范的“error”概念。例如，一个带有有效性验证的解析器会使用这个方法来报告违反有效性验证或违背有效性约束的情况。对于该类错误默认是什么也不做。SAX 解析器调用这个方法后可继续提供正常的解析事件。
- fatalError()：该方法对应于 XML1.0 规范的“fatal error”概念。例如，一个解析器会使用这个方法来报告违反格式良好约束的情况。一旦在该方法被调用，SAX 解析器可以停止正常的解析事件。

下述代码用于实现任务【描述 6.D.3】，创建解析程序，演示 SAX 错误处理的应用。

【描述 6.D.3】 SAXErrorHandler.java

```
import java.io.File;
import java.io.IOException;
import javax.xml.parsers.SAXParser;
import javax.xml.parsers.SAXParserFactory;
import org.xml.sax.SAXException;
import org.xml.sax.SAXParseException;
import org.xml.sax.helpers.DefaultHandler;
public class SAXErrorHandler extends DefaultHandler {
    // 覆盖 warning()方法
    public void warning(SAXParseException exception) {
        System.out.println("*******WARNING******");
        System.out.println("\t 行:\t" + exception.getLineNumber());
        System.out.println("\t 列:\t" + exception.getColumnNumber());
        System.out.println("\t 错误信息:\t" + exception.getMessage());
        System.out.println("**********************");
    }
    // 覆盖 error()方法
    public void error(SAXParseException exception) throws SAXException {
        System.out.println("******* ERROR ******");
        System.out.println("\t 行:\t" + exception.getLineNumber());
        System.out.println("\t 列:\t" + exception.getColumnNumber());
        System.out.println("\t 错误信息:\t" + exception.getMessage());
        System.out.println("**********************");
    }
    // 覆盖 fatalError()方法
    public void fatalError(SAXParseException exception) throws SAXException {
        System.out.println("******** FATAL ERROR ********");
        System.out.println("\t 行:\t" + exception.getLineNumber());
```

```
        System.out.println("\t 列:\t" + exception.getColumnNumber());
        System.out.println("\t 错误信息:\t" + exception.getMessage());
        System.out.println("**************************");
    }
    public static void main(String[] args) {
        String xmlFilename = "SAXErrorHandler.xml";
        // 创建 SAX 解释器工厂对象
        SAXParserFactory spf = SAXParserFactory.newInstance();
        SAXParser saxParser = null;
        try {
            // 通过工厂对象创建 SAX 解释器 SAXParser 对象
            saxParser = spf.newSAXParser();
        } catch (Exception ex) {
            System.err.println(ex);
        }
        try {
            // 使用 SAX 解释器加载 XML 文档,把实现了 DefaultHander 的实例对象装入解释器中
            saxParser.parse(new File(xmlFilename), new SAXErrorHandler());
        } catch (SAXException se) {
            System.err.println(se.getMessage());
        } catch (IOException ioe) {
            System.err.println(ioe);
        }
    }
}
```

上述程序继承 DefaultHandler，但只对 ErrorHandler 接口的方法进行重写，在每个方法中输出的信息包括出错的行、列及错误信息。给定如下 XML 文档，执行上述程序没有错误信息输出。

【描述 6.D.3】 SAXErrorHandler.xml

```
<?xml version="1.0" encoding="GB2312"?>
<图书列表>
    <技术书籍>
        <图书>
            <书名>数据库原理与应用</书名>
            <价格 货币单位="RMB">59</价格>
            <出版日期>2010,7,24</出版日期>
        </图书>
    </技术书籍>
</图书列表>
```

为演示错误处理，人为制造一些错误来检查一下错误处理器工作情况。删除元素<出版

日期>的闭合标记，再次执行上述程序，解析过程会产生一个 fatal error，下面是执行结果：

```
******** FATAL ERROR ********
    行: 8
    列: 11
    错误信息:The element type "出版日期" must be terminated by the matching end-tag
"</出版日期>".
*****************************
The element type "出版日期" must be terminated by the matching end-tag "</出版日
期>".
```

4. 过滤器

在 SAX API 中还提供了一个过滤器接口 XMLFilter，以及对它的默认实现 XMLFilterImpl。使用该接口（类）可以很容易地开发出复杂的 SAX 应用。在应用程序中，开发者可以构造多个过滤器，每个过滤器实现一个特定功能，然后将它们串成过滤器链，从而创建结构复杂的应用程序。

在 SAX API 中，XMLFilter 接口继承自 XMLReader 接口，与 XMLReader 不同的是它不像 XMLReader 那样通过解析文档来获取事件，而是从其他 XMLReader 中获取事件，当然这也包括从其他的 XMLFilter 中获取事件。在 XMLFilter 中有以下两个方法。

- setParent(XMLReader parent)：设置父 XMLReader，该方法让应用程序将这个过滤器连接到它的父 XMLReader（也可能是另一个过滤器）。
- XMLReader getParent()：获取父 XMLReader，该方法让应用程序可以查询父 XMLReader（也可能是另一个过滤器）。

开发者不需要自己实现 XMLFilter 接口，在 SAX API 中提供了一个 XMLFilterImpl 类，它不仅实现了 XMLFilter 接口而且还实现了其他四个核心处理器接口，开发者只需要继承它即可。

下述代码用于实现任务【描述 6.D.4】，结合现有实例，构造过滤器，通过过滤掉指定 XML 文档中的元素来演示 XMLFilter 的作用。

首先定义过滤器类，该过滤器在开始标签、结束标签及文本处理方法内部，都对处理的内容进行判断，只要是“<技术书籍>”元素，都不予处理。

【描述 6.D.4】 MyFilter.java

```
import org.xml.sax.*;
import org.xml.sax.helpers.*;
public class MyFilter extends XMLFilterImpl {
    // 记录当前元素的名称
    private String currentElement;
    private String elementName="技术书籍";
    // 构造过滤器
```

```
    public MyFilter(XMLReader parent) {
        super(parent);
    }
    // 过滤掉元素<技术书籍>的开始事件
    public void startElement(String namespaceURI, String localName,
            String fullName, Attributes attributes) throws SAXException {
        currentElement = fullName;
        // 如果该元素不是<技术书籍>元素，交给事件处理程序处理，否则不做任何处理
        if (!fullName.equals(elementName)) {
            super.startElement(namespaceURI, localName, fullName, attributes);
        }
    }
    // 过滤掉元素<技术书籍>的结束事件
    public void endElement(String namespaceURI, String localName,
            String fullName) throws SAXException {
        // 如果该元素不是<技术书籍>元素，交给事件处理程序处理，否则不做任何处理
        if (!fullName.equals(elementName)) {
            super.endElement(namespaceURI, localName, fullName);
        }
    }
    // 过滤掉元素<技术书籍>中的内容
    public void characters(char[] buffer, int start, int length)
            throws SAXException {
        // 如果不是<技术书籍>元素的文本内容，交给事件处理程序处理，否则跳过
        if (!currentElement.equals(elementName)) {
            super.characters(buffer, start, length);
        }
    }
}
```

上述代码通过继承 XMLFilterImpl 类实现的那些事件处理方法，如 startElement()、endElement()、characters()等是该过滤器实现它自身功能的地方。而通过继承 XMLFilterImpl 而实现的这个类会被作为成各种处理器（它本身实现了四个处理器接口）用在它的父 XMLReader 中。该步骤会在调用过滤器的 parse()方法时被自动执行。

创建 SAX 处理程序，代码如下所示。

【描述 6.D.4】 MyFilterHandler.xml

```
import java.io.IOException;
import javax.xml.parsers.SAXParser;
import javax.xml.parsers.SAXParserFactory;
import org.xml.sax.*;
import org.xml.sax.helpers.DefaultHandler;
```

```
public class MyFilterHandler extends DefaultHandler {
    // 存储文本内容
    private StringBuffer buf;
    public void startDocument() throws SAXException {
        buf = new StringBuffer();
        System.out.println("*******开始解析文档*******");
    }
    public void endDocument() throws SAXException {
        System.out.println("*******解析文档结束*******");
    }
    public void startElement(String namespaceURI, String localName,
            String fullName, Attributes attributes) throws SAXException {
        System.out.println("元素: " + "[" + fullName + "]" + " 开始解析!");
        // 打印出属性信息
        for (int i = 0; i < attributes.getLength(); i++) {
            System.out.println("\t 属性名称:" + attributes.getLocalName(i)
            + " 属性值:" + attributes.getValue(i));
        }
    }
    public void endElement(String namespaceURI, String localName,
            String fullName) throws SAXException {
        // 打印出非空的元素内容并将 StringBuffer 清空
        String nullStr = "";
        if (!buf.toString().trim().equals(nullStr)) {
            System.out.println("\t 内容是: " + buf.toString().trim());
        }
        buf.setLength(0);
        // 打印元素解析结束信息
        System.out.println("元素: " + "[" + fullName + "]" + " 解析结束!");
    }
    public void characters(char[] chars, int start, int length)
            throws SAXException {
        // 将元素内容累加到 StringBuffer 中
        buf.append(chars, start, length);
    }
    public static void main(String[] args) {
        String xmlFilename = " SAXErrorHandler.xml";
        // 创建 SAX 解释器工厂对象
        SAXParserFactory spf = SAXParserFactory.newInstance();
        SAXParser saxParser = null;
        XMLReader reader = null;
        try {
            // 通过工厂对象创建 SAX 解释器 SAXParser 对象
```

```
            saxParser = spf.newSAXParser();
            // 获取 XMLReader 实例
            reader = saxParser.getXMLReader();
            // 初始化过滤器
            XMLFilter myFilter = new MyFilter(reader);
            // 创建 DefaultHandler 的实例
            DefaultHandler defaultHandler = new MyFilterHandler();
            // 为过滤后的事件流设置 ContentHandler
            myFilter.setContentHandler(defaultHandler);
            // 为过滤后的事件流设置 ErrorHandler
            myFilter.setErrorHandler(defaultHandler);
            // 开始解析文档，注意是使用 myFilter 中的解析方法
            myFilter.parse(xmlFilename);
        } catch (IOException e) {
            System.out.println("读入文档时错: " + e.getMessage());
        } catch (SAXException e) {
            System.out.println("解析文档时错: " + e.getMessage());
        } catch (Exception e) {
            System.out.println(e);
        }
    }
}
```

上述代码中，SAX 基础事件处理功能与问题【描述 6.D.1】一致。需要强调的是过滤器的构造过程，首先需要通过 SAX 解析器实例对象获取其封装的 XMLReader 对象（reader），然后通过构造方法将 XMLReader 对象指派给过滤器对象（myFilter = new MyFilter(reader)），当执行过滤器的 parse()方法时，该过滤器奏效；另外，在继承 XMLFilterImpl 的类中用 setXXX() 方法注册的事件处理器是给过滤后的事件流而用的。

使用上述程序分析问题【描述 6.D.3】的 XML 文档 SAXErrorHandler.xml，执行结果如下：

```
*******开始解析文档*******
元素：[图书列表] 开始解析!
元素：[图书] 开始解析!
元素：[书名] 开始解析!
    内容是：数据库原理与应用
元素：[书名] 解析结束!
元素：[价格] 开始解析!
    属性名称:货币单位 属性值:RMB
    内容是：59
元素：[价格] 解析结束!
元素：[出版日期] 开始解析!
    内容是：2010,7,24
元素：[出版日期] 解析结束!
```

```
元素：[图书] 解析结束!
元素：[图书列表] 解析结束!
*******解析文档结束*******
```

通过运行结果可以发现有关<技术书籍>的全部事件已经被过滤掉了。仔细分析一下可以发现，<技术书籍>元素的子元素依然存在。这是因为，SAX 是把 XML 文档解析成事件流，所有没有被过滤的事件都会保留下来，这也是 SAX 和 DOM 的最大不同（在 DOM 中文档被解析成树状模型，如果删除一个元素，那么这个元素及它的子元素就都会被删除，这符合树状模型的特点）。

注意 如果不是使用带参数的构造器创建 XMLFilter 对象，务必使用 setParent(XMLReader parent)方法连接它的父 XMLReader; 如果使用多个过滤器的话，执行顺序是从父亲到最后的过滤器。

6.2 使用 JDOM 解析 XML

JDOM 是一个开源的、专为 Java 语言提供 XML 解析功能的项目，它基于树形结构，利用纯 Java 的技术对 XML 文档实现解析、生成、序列化以及多种操作，JDOM 直接为 Java 编程服务。它利用更为强有力的 Java 语言的诸多特性（方法重载、集合概念以及映射），把 SAX 和 DOM 的功能有效地结合起来。

6.2.1 JDOM 概述

JDOM 在 2000 年被 Brett McLaughlin 和 Jason Hunter 开发出来，以弥补 DOM 及 SAX 在实际应用中不足之处。这些不足之处主要在于 SAX 没有文档修改、随机访问以及输出的功能；而对于 DOM 来说，DOM 是一个接口定义语言（IDL），它的任务是在不同语言实现中的一个最低的通用标准，并不是为 Java 特别设计的。

JDOM 几乎能够实现 org.w3c.dom 包所定义的所有功能，但它并不是一个简单的替代品，JDOM 能够与 SAX 和 DOM 很好地结合。JDOM 建立在现有的 API 的能力之上，它提供的类的封装从配置和运行分析器执行中分担了大量工作，可以解析 XML 文档，可以按照 Schema 或者 DTD 规则文件修改文档结构和内容，并且可以把 XML 文档的内容提供给应用程序，但它不负责根据文本输入来对 XML 进行语法分析。JDOM 已经被收录到 JSR-102 内，这标志着 JDOM 成为了 Java 平台组成的一部分。

DOM 和 SAX 是解析 XML 的底层的 API，各个厂商、组织的 XML 解析器都是在使用其中的一种或者两种 API 的基础上，向用户提供更方便的接口。如图 6-3 所示是 JDOM 的结构图，可以看出，当输入 XML 文档时，JDOM 使用 SAXBuilder，当输入为 DOM 树的时候，才用 DOMBuilder。

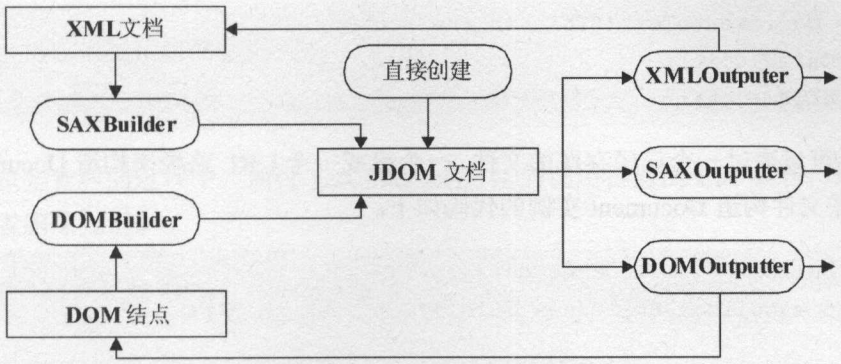


图 6-3　JDOM 结构图

6.2.2　JDOM 的 API

JDOM 中的常用类主要包括 SAXBuilder、DOMBuilder、Document、XMLOutputter、Element 和 Attribute。

1. SAXBuilder

JDOM 中的 SAXBuilder 类会使用 SAX 来建立一个 JDOM 的解析树。它可以通过 build() 方法由指定的输入数据流建立一个文件，返回一个 Document 对象。

2. DOMBuilder

JDOM 中的 DOMBuilder 类会使用 DOM 来建立一个 JDOM 的解析树。它可以通过 build() 方法由指定的输入数据流建立一个文件，返回一个 Document 对象。

3. Document

Document 类的一个实例用来描述一个 XML 文档。这个文档类是轻量级的，它可以包括文档类型、处理指令对象、根元素和注释对象等内容。

例如，通过如下代码将构造一个包含根元素的 Document 对象：

```
Element root = new Element("GREETING");
Document doc = new Document(root);
root.setText("Hello JDOM!");
```

或者简单地使用如下代码声明：

```
Document doc = new Document(new Element("GREETING").setText("Hello JDOM!t"));
```

如果使用 DOM 完成相同的操作，则需要更为复杂的代码：

```
DocumentBuilderFactory factory =DocumentBuilderFactory.newInstance();
DocumentBuilder builder =factory.newDocumentBuilder();
Document doc = builder.newDocument();
Element root =doc.createElement("root");
```

```
Text text = doc.createText("This is the root");
root.appendChild(text);
doc.appendChild(root);
```

另外，还可以通过一个已经存在的文件、一个流或一个 URL 路径来构造 Document 实例。

通过存在文件构造 Document 实例的代码如下：

```
DOMBuilder builder = new DOMBuilder();
Document doc = builder.build(new File("jdom_test.xml"));
```

通过 URL 构造 Document 实例的代码如下：

```
SAXBuilder builder = new SAXBuilder();
Document doc = builder.build(url);//url 代表路径
```

通过流构造 Document 实例的代码如下：

```
SAXBuilder builder = new SAXBuilder();
Document doc = null;
Reader in= new StringReader(textXml);
doc = builder.build(in);
```

4. XMLOutPutter

一个 XML 文档可以以多种格式输出，在 JDOM 中最常用的是字节流。XMLOutputter 类提供了这种特性，它将 XML 文档写入一个特定的 OutputStream 流中。代码如下：

```
XMLOutputter outer=new XMLOutputter();
outer.output(doc,System.out);
```

其中 output()方法用于将 XML 文档输出到指定的位置，既可以是标准的输出 System.out，也可以是 File 对象或者 OutputStream 对象。

5. Element

在 JDOM 中，XML 元素就是 Element 类的实例。Element 类有两种主要的操作，一个是浏览元素树，另一个是移动元素。

JDOM 可以很容易地获到根元素，代码如下：

```
Element root = doc.getRootElement();
```

并获得根元素的所有子元素的一个列表：

```
List allChildren = root.getChildren();
```

还可以通过名字获得指定的子元素列表：

```
List namedChildren = root.getChildren("Jack");
```

此外，可以根据指定名称得到第一个元素：

```
Element child = root.getChild("Jack");
```

在浏览元素树的过程中，通过 getChildren()方法的调用将返回 List 集合，用户可以在其中添加或者删除元素。

```
List allChildren = root.getChildren();
//删除第四个子元素
allChildren.remove(3);
// 删除所有名称为 jack 的子元素
allChildren.removeAll(root.getChildren("Jack"));
//在集合末尾添加新元素
allChildren.add(new Element("Jack"));
//在集合开头添加新元素
allChildren.add(0,new Element("Tom"));
```

JDOM 可以很方便地在文档内部或者在文档之间移动元素。代码如下：

```
Element movable = new Element("test");
parent1.addContent(movable);//添加
parent1.removeContent(movable);//删除
```

6. Attribute 类

可以使用 Element 类的 getAttribute()方法来取得一个元素的属性，该方法会返回一个 Attribute 对象。Attribute 类提供了 getValue()方法，它将会以字符串的形式返回一个属性值。例如通过下面代码可以得到 name 元素的属性值：

```
element.getAttribute("name").getValue();
```

如果想在一个元素上新增一个属性，示例代码如下：

```
Attribute att=new Attribute("name","Berry");
Element.addAttribute(att);
```

6.2.3　JDOM 编程

JDOM 使用标准的 Java 编码模式进行编程，尽量使用 new 操作符而不是工厂方法来获取类的实例，因此对于熟悉 Java 的用户来说使用 JDOM 是非常方便的。

JDOM 与 DOM 和 SAX 不同，它不是 JDK 中自带的包，用户需要访问 JDOM 项目站点 http://jdom.org，并下载最新版本的 JDOM 工具包，然后解压缩。JDOM 所需的 jar 文件就是 build 目录下的 jdom.jar 文件，将其添加到 CLASSPATH 环境变量中，这样就可以在 Java 中使用 JDOM 了。

1. 创建 XML 文档

下述代码用于实现任务【描述 6.D.5】，通过 JDOM 创建一个 XML 文档，并将其保存在本地磁盘上。

【描述 6.D.5】 JDOMCreate.java

```
import org.jdom.*;
import org.jdom.output.*;
import java.io.*;
public class JDOMCreate {
    public void createXML() throws Exception {
        // 声明 XML 文档中的各个元素
        Element root, student, name, age;
        // 创建根元素
        root = new Element("student-info");
        // 创建各个子元素
        student = new Element("student");
        // 增加 id 属性
        student.setAttribute("id", "95001");
        name = new Element("name");
        age = new Element("age");
        // 将根元素植入文档 doc 中
        Document doc = new Document(root);
        // 设置各个子元素的内容
        name.setText("李明");
        age.setText("24");
        // 将 name、age 子元素添加到 student 元素中
        student.addContent(name);
        student.addContent(age);
        // 将 student 元素添加到根元素中
        root.addContent(student);
        // 声明文档格式对象
        Format format = Format.getCompactFormat();
        // 设置 xml 文件的字符为 GB2312
        format.setEncoding("GB2312");
        // 设置 xml 文件的缩进为 3 个空格
        format.setIndent("   ");
        // 将格式应用到输出流中
        XMLOutputter XMLOut = new XMLOutputter(format);
        // 将文档通过文件输出流生成 studentinfo.xml 文件
        XMLOut.output(doc, new FileOutputStream("students.xml"));
    }
    public static void main(String[] args) throws Exception {
```

```
        JDOMCreate jdom = new JDOMCreate();
        System.out.println("正在生成 XML 文档.....");
        jdom.createXML();
        System.out.println("完成");
    }
}
```

执行上述程序，将在项目根目录下生成一个新的 XML 文档，名称为 students.xml，该 XML 文档内容如下：

```
<?xml version="1.0" encoding="GB2312"?>
<student-info>
  <student id="95001">
    <name>李明</name>
    <age>24</age>
  </student>
</student-info>
```

2. 遍历 XML 文档

下述代码用于实现任务【描述 6.D.6】，使用 JDOM 载入给定的 XML 文档，然后遍历并显示 XML 文档中的全部元素。

【描述 6.D.6】 JDOMRead.java

```
import java.io.File;
import java.util.List;
import org.jdom.Attribute;
import org.jdom.Document;
import org.jdom.Element;
import org.jdom.input.SAXBuilder;
public class JDOMRead {
    // 缓冲字符串，用于存储读取到的 XML 信息
    private static StringBuffer strbuf = new StringBuffer();
    public static void main(String[] argv) {
        String filename = "students.xml";
        try {
            SAXBuilder sb = new SAXBuilder();
            // 使用文件名直接构造 Document 对象
            Document doc = sb.build(new File(filename));
            // 获取根元素
            Element root = doc.getRootElement();
            strbuf.append("<?xml version=\"1.0\" encoding=\"GB2312\"?>\n");
            strbuf.append("<");
            strbuf.append(root.getName());
```

```
            // 获取根元素的属性
            accessAttribute(root);
            strbuf.append(">\n");
            // 遍历结点
            accessElement(root);
            strbuf.append("</" + root.getName() + ">");
            // 输出缓冲内容
            System.out.println(strbuf.toString());
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
    // 遍历结点
    public static void accessElement(Element parent) {
        List listChild = parent.getChildren();
        int iChild = listChild.size();
        for (int i = 0; i < iChild; i++) {
            Element e = (Element) listChild.get(i);
            strbuf.append("<" + e.getName());
            // 访问结点属性
            accessAttribute(e);
            strbuf.append(">\n");
            // 访问结点文本值
            if (e.getTextTrim() != null && e.getTextTrim().length() != 0) {
                strbuf.append(e.getTextTrim() + "\n");
            }
            // 递归访问结点
            accessElement(e);
            strbuf.append("</" + e.getName() + ">\n");
        }
    }
    // 访问结点的属性
    public static void accessAttribute(Element e) {
        // 获取结点属性列表
        List listAttributes = e.getAttributes();
        int iAttributes = listAttributes.size();
        // 循环输出属性信息
        for (int j = 0; j < iAttributes; j++) {
            Attribute attribute = (Attribute) listAttributes.get(j);
            strbuf.append(" " + attribute.getName() + "=\""
                    + attribute.getValue() + "\"");
        }
    }
}
```

上述程序使用递归的方式对输入的 XML 文档进行解析，引入任务【描述 6.D.5】中生成的 XML 文档，执行该程序，结果如下：

```
<?xml version="1.0" encoding="GB2312"?>
<student-info>
<student id="95001">
<name>
李明
</name>
<age>
24
</age>
</student>
</student-info>
```

JDOM 解析也支持命名空间，可借助于 Namespace 类获取。另外，通过 JDOM 也可很方便地实现元素的查找、删除、修改功能，在此不再赘述。

6.3　SAX、DOM 和 JDOM 技术的比较

在了解了 SAX、DOM 和 JDOM 的原理及解析 XML 文档的方法后，对这三种解析技术进行比较，明确其优缺点，这样在实际开发中就可以根据实际的开发环境而选择合适的解析技术。SAX、DOM 和 JDOM 的优缺点如表 6-2 所示。

表 6-2　SAX、DOM 和 JDOM 技术的优缺点比较

名称	优点	缺点	适用场合
SAX	① 无须将整个文档加载到内存，所以内存消耗少 ② 推模型允许注册多个 ContentHandler	① 没有内置的文档导航支持 ② 不能够随机访问 XML 文档 ③ 不支持在原地修改 XML ④ 不支持名字空间作用域	适用于只从 XML 文档读取数据的应用程序（不可用于操作或修改 XML 文档）
DOM	① 易于使用 ② 丰富的 ASI 集合，可用于轻松的导航 ③ 整棵树加载到内存，允许对 XML 文档进行随机访问	① 整个 XML 文档必须一次解析完 ② 将整棵树加载到内存中成本比较高 ③ 一般的 DOM 结点对于必须为所有结点创建对象的对象类型绑定不太理想	适用于需要修改 XML 文档的应用程序或 XSLT 应用程序（不可用于只读 XML 的应用程序）

（续表）

名称	优点	缺点	适用场合
JDOM	① 基于树的处理XML的Java API，把树加载在内存中 ② 没有向下兼容的限制，因此比DOM简单 ③ 速度快，缺陷少 ④ 具有SAX的Java规则	① 不能处理大于内存的文档 ② DOM表示XML文档逻辑模型，不能保证每个字节真正变换 ③ 针对实例文档不提供DTD与模式的任何实际模型 ④ 不支持与DOM中相应遍历包	适用于既具有树的遍历，也有SAX的Java规则，在需要平衡时使用

注意 DOM API的另一个不易察觉的问题是：使用它写成的代码要扫描XML文档两次。第一次将DOM结构读进内存，第二次定位感兴趣的数据。相反，SAX编程模式支持一趟同时定位和收集XML数据。

小结

通过本章的学习，学生应该能够学会：

- SAX是Simple API for XML的缩写。
- SAX不是W3C的推荐标准，是开源的产物。
- SAX是基于事件驱动的、轻量型的XML处理方法。
- SAX基于流式处理，不需要读入整个文档，而是边解析边处理。
- DOM和SAX其实都是与语言无关的，并非Java所独有，只要有相应的语言实现，DOM和SAX可以应用在任何面向对象的语言中。
- SAX解析器中主要有5种事件：startDocument、startElement、characters、endElement、endDocument。
- SAXParserFactory类是SAX解析器工厂类，开发者通过该工厂类实例来获得一个具体的解析器对象（SAXParser）。
- DefaultHandler是SAX事件的核心处理类，该类实现了四个核心SAX2处理程器接口。
- ErrorHandler接口是SAX的错误处理接口。
- SAX中，开发者可以通过实现XMLFilter接口进行解析过滤，并可通过多个过滤器类相互“封装”形成过滤器链。
- JDOM是一个开源的、专为Java语言提供XML解析功能的项目。
- JDOM集成了DOM和SAX处理的特点，是但它并不是一个简单的替代品。
- JDOM中的常用类主要包括SAXBuilder、DOMBuilder、Document、XMLOutputter、Element和Attribute等。
- JDOM不是JDK中自带的包，需要单独下载，并添加到类路径中才能使用。

练习

1. 下述关于 SAX 的描述，错误的是_____。

   A. SAX 的一个 W3C 标准

   B. SAX 是 Simple API for XML 的缩写

   C. SAX 并不需要读入整个 XML 文档

   D. SAX 是基于事件驱动的，适于处理大型 XML 文件

2. 下述关于 SAX 与 DOM 的比较，错误的是_____。

   A. DOM 是基于对象的，SAX 是基于流式的

   B. DOM 需要读入整个 XML 文档才能处理，SAX 是边解析边处理

   C. 相对于 DOM，SAX 适于处理大型的 XML 文件

   D. DOM 是语言无关的，而 SAX 的 Java 特有的

3. 下述关于 SAX 事件的描述，正确的是_____。

   A. startDocument 事件表明 SAX 解析器发现了文档的开始

   B. endDocument 事件表明 SAX 解析器发现 XML 文档的结尾

   C. startElement 事件表明 SAX 解析器发现 XML 文件中一个元素的起始标签

   D. characters 事件表明 SAX 解析器发现 XML 文档中一个元素的文本信息

4. _____类是 SAX 解析器工厂类，以便利用这个工厂类来获得一个具体的_____类对象（解析器类对象），_____类是 SAX 事件的回调处理类。

5. 对于下列 XML 文档：

```
<?xml version="1.0" encoding="GB2312"?>
<Orders>
    <Order orderID="A001" orderDate="2010-7-20">
        <name>玩具</name>
        <number>16</number>
        <city>上海</city>
        <zip>200000</zip>
        <phoneno>13577778888</phoneno>
    </Order>
    <Order orderID="A002" orderDate="2010-7-22">
        <name>文具</name>
        <number>17</number>
        <city>青岛</city>
        <zip>266000</zip>
        <phoneno>0532-66667777</phoneno>
    </Order>
    <Order orderID="A004" orderDate="2010-8-22">
        <name>衣物</name>
```

```
        <number>56</number>
        <city>青岛</city>
        <zip>266000</zip>
        <phoneno>0532-88889999</phoneno>
    </Order>
</Orders>
```

使用 JDOM，创建一个 XML 处理程序，要求实现如下功能：

- 载入并遍历给定 XML 文档的内容。
- 能够根据用户输入的订单号，查找相应的元素，并显示其信息。
- 将内存中的最新数据写入到新文件中。
- 上述功能要求使用菜单驱动方式实现。

拓展练习：基于 SAX，创建 XML 处理程序实现对上述 XML 文档的遍历（提示：使用栈结构）。

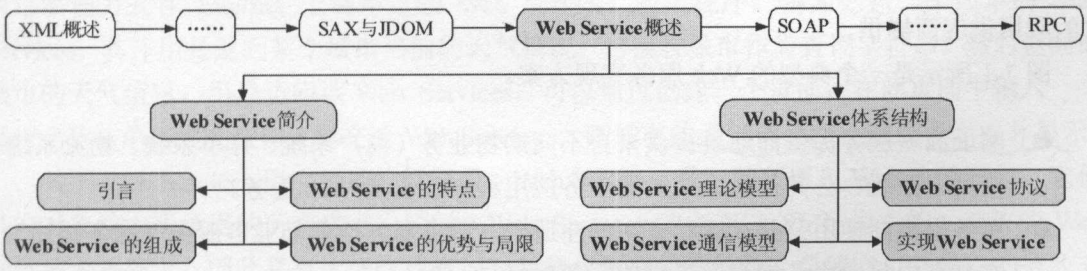
6. 请描述使用 DOM 和 SAX 处理 XML 的区别。

第 7 章　Web Services 概述

本章目标

- 了解 Web Services 的应用背景
- 了解 Web Services 的特点及功能
- 掌握 Web Services 的技术组成
- 了解 Web Services 的优势和局限
- 掌握 Web Services 的工作原理
- 掌握 Web Services 的协议构成
- 了解 Web Services 的通信模型
- 掌握实现 Web Services 的开发步骤
- 了解常用的 Web Services 开发平台

学习导航



7.1 Web Services 简介

事实证明 Java 是一种非常成功的编程语言和应用平台，XML 出现后一直以惊人的速度发展，它与 Java 一样也获得了巨大的成功。如果将 XML 与 Java 结合，Internet 数据的集成度将远远超过单独使用 Java 时的集成度。近些年来，Web Services 一直是程序设计人员研究的热点之一，它的发展将极大地推动企业内和企业间的应用集成，提高应用程序的互操作性。

7.1.1 引言

Web Services（Web 服务）是一种分布式的计算技术，是符合业界标准的分布式应用组件，能够基于开放的标准和技术在 Internet 上实现应用程序之间的互操作，Web Services 在网络中通过标准的 XML 协议和消息格式来发布和访问商业应用服务。通过 Web Services，可以在 Web 站点放置可编程的元素，发布能满足特定功能的在线应用服务，并可以使用各种计算平台、手持设备、家用电器对其进行动态查找、订阅和访问，极大地拓展了应用程序的功能，实现了软件的动态提供。

Web Services 是建立于 SOA（Service-Oriented Architecture，面向服务的体系结构）基础之上的最新分布式计算技术，可以将软件组件（包括来自不同系统的对象、函数等）发布为服务。简而言之，Web Services 是自描述的模块化业务程序，其通过可编程接口经 Internet 将业务逻辑发布为服务，并通过 Internet 协议来动态查找、订阅和使用这些服务。

Web Services 建立在 XML 标准上，可以使用任何编程语言、协议或平台开发出松散耦合的应用，让任何人能够在任何时间通过任何平台访问该业务程序。图 7-1 显示了一个 Web Services 应用的简单示例，其中网上商城服务提供商将其业务应用程序发布为支持顾客和应用程序客户端的 Web 服务，而这些业务应用程序可能是由处于不同平台、不同网络或地理位置的应用供应商提供。

图 7-1 所示是一个典型的 Web 服务实现方案。

- 网上商城服务提供商通过提供来自不同购物业务（商户系统、订单系统、物流系统、信用卡系统、公共信息系统）的业务应用程序部署其 Web 服务。
- 服务提供商使用公共（或私有）的注册表（服务器）注册其业务服务（服务描述）。注册表中包含服务提供商提供的服务信息。
- 客户可使用各种平台或设备（手机、计算机、各种终端、家电设备等），通过 Internet 或其他网络途径查找服务注册表来找到相应的 Web 服务，然后调用该服务的功能。

该示例通过图示的形式说明了如何将一个较为复杂的业务功能发布为 Web 服务，以及客户如何调用这些服务。

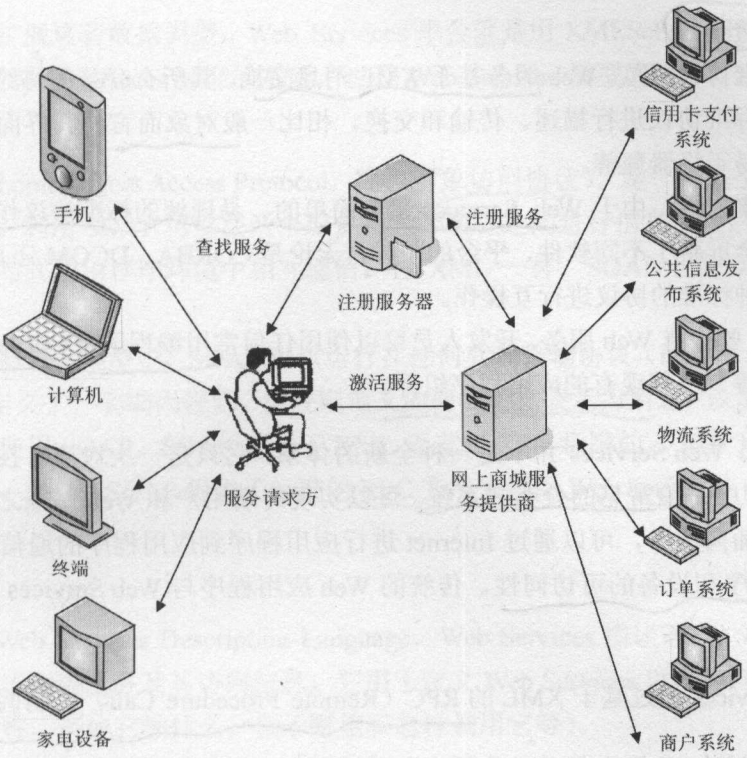


图 7-1　Web Services 应用示例

7.1.2　Web Services 的特点

Web Services 就是一个向外界暴露出的能够通过 Internet 进行调用的应用程序，用户可以通过编程方式在 Internet 上调用这些 Web Services 应用程序。例如创建一个简单的 Web Services，其作用是返回某个城市当前的天气情况：它接收城市作为查询字符串，然后返回该城市的天气信息。用户访问该 Web Services，可以通过创建一个页面，在此页面中输入一个城市名称，单击提交按钮（提交给 Web Service），则城市的天气信息将返回到页面中并显示。

关于 Web Services 更为精确的解释是：Web Services 是一种部署在 Web 上的对象，它们具有对象技术所承诺的所有优点，同时，Web Services 建立在以 XML 为主的、开放的 Web 规范技术基础上，因此具有比任何现有的对象技术更好的开放性，是建立可互操作的分布式应用程序的新平台。Web Services 平台是一套标准，它定义了应用程序如何在 Web 上实现互操作性，用户可以用任何语言，在任何平台上编写所需要的 Web Services。对于外部的 Web Services 使用者而言，Web Services 实际上是一种部署在 Web 上的对象或者组件，其业务逻辑对使用者来说是透明的。Web Services 应用程序具备如下特征。

- 封装性。Web Services 是一种部署在 Web 上的对象，具备对象的良好封装性，而对于使用者而言，仅能看到该对象提供的功能列表。
- 松散耦合。只要 Web Services 的调用接口不变，Web Services 的内部变更对调用者来

说都是透明的。

- 使用标准协议规范。Web 服务基于 XML 消息交换，其所有公共的协约完全需要使用开放的标准协议进行描述、传输和交换。相比一般对象而言，其界面调用更加规范化，更易于机器理解。
- 高度可集成性。由于 Web Services 采取简单的、易理解的标准协议作为组件描述，所以完全屏蔽了不同软件、平台的差异，无论是 CORBA、DCOM 还是 J2EE 都可以通过这种标准的协议进行互操作。
- 易构建。要构建 Web 服务，开发人员可以使用任何常用编程语言（如 Java、C#、C/C++ 或 Perl 等）及其现有的应用程序组件。

从本质上看，Web Services 并不是一种全新的体系，它只是一次对原有技术的革新。早期的 Web 应用程序是最常见的分布式系统，可以实现终端用户和 Web 站点之间的交互，而 Web Services 则面向服务，可以通过 Internet 进行应用程序到应用程序的通信，并提供不同环境下的应用程序和设备的可访问性。传统的 Web 应用程序与 Web Services 之间有着显著的区别。

- Web Services 通过基于 XML 的 RPC（Remote Procedure Call）机制调用，可以穿越防火墙。
- Web Services 可以提供基于 XML 消息交换的、跨平台、跨语言的解决方案。
- Web Services 基于轻量级构建，可简化应用程序集成。
- Web Services 可方便地实现异构应用程序间的互操作。

7.1.3 Web Services 的组成

Web Services 平台提供了一套标准的类型系统，以用于沟通不同平台、编程语言和组件模型中的数据类型。在传统的分布式系统平台中，提供了一些方法来描述界面、方法和参数（如：COM 和 CORBA 中的 IDL 语言），同样，在 Web Services 平台中也提供了一种标准来描述这些 Web Services，使得客户可以得到足够的信息来调用这些 Web Services；此外，还提供了一种方法来对这些 Web Services 进行远程调用，这种方法实际上是一种远程过程调用协议（RPC），为了达到互操作性，这种 RPC 协议必须与平台和编程语言无关。

从总体上说来，用于构建和使用 Web 服务主要有四种标准和技术：XML、SOAP、WSDL、和 UDDI。

1. XML

XML 是 Web Services 平台中表示数据的基本格式，XML 使用 Unicode 编码，采用自描述的数据结构，能够以简单的文本文档格式存储、传输、读取数据。现在 XML 已经作为应用程序、系统和设备之间通过 Internet 交换信息的通用语言而被广泛接受。

另外，W3C 制定了一套标准——XMLSchema，它定义了一套标准的数据类型，并给出

了一种语言来扩展这套数据类型，Web Services 平台就是用 XMLSchema 作为其数据类型系统的。XML 是 Web Services 标准的基础，也是 Web Services 模型的核心。

2. SOAP

SOAP（Simple Object Access Protocol，简单对象访问协议），是一种基于 XML 的轻量级消息交换协议。利用 SOAP 可以在两个或多个对等实体之间进行信息交换，并可以使这些实体在分散的分布式应用程序环境中相互通信。与 XML 一样，SOAP 也独立于语言、运行平台或设备。

在 Web Services 模型中，SOAP 可以运行在任何其他传输协议（HTTP、SMTP、FTP 等）之上。SOAP 定义了一套编码规则，该规则定义如何将数据表示为消息，以及怎样通过 HTTP 等传输协议来使用 SOAP，SOAP 是基于 XML 语言和 XSD 标准的，其中 XML 是 SOAP 的数据表示方式。另外，SOAP 提供了标准的 RPC 方法来调用 Web Services，以请求/响应模型运行。

3. WSDL

WSDL（Web Services Description Language，Web Services 描述语言）标准是一种 XML 格式，用于描述网络服务及其访问信息。它用于定义 Web Services 以及如何调用它们（描述 Web 服务的属性，例如它做什么，位于哪里和怎样调用它等）。

在 Web Services 模型中，WSDL 用于定义 Web 服务的元数据语言，描述服务提供方和请求方之间如何进行通信。WSDL 文档可用于动态发布 Web Services 功能、查找已发布的 Web Services 以及绑定 Web Services。在 WSDL 中包含了使用 SOAP 的服务描述的绑定，也包含了使用简单 HTTP GET 和 POST 请求的服务描述的绑定。

4. UDDI

UDDI（Universal Description，Discovery and Integration）是通用描述、发现和集成的英文缩写，它是由 Ariba、IBM、微软等公司倡导的，它提供了在 Web 上描述并发现商业服务的框架。UDDI 定义了一种在通用注册表中注册 Web 服务并划分其类别的机制。查询 UDDI 注册表以寻找某项服务时，将返回描述该服务接口的 WSDL 描述。通过 WSDL 描述，开发人员可以开发出与服务提供方通信的 SOAP 客户端接口。

UDDI 的核心组件是 UDDI 商业注册，它使用一个 XML 文档来描述企业及其提供的 Web 服务。从概念上来说，UDDI 商业注册所提供的信息包含三个部分："白页（White Page）"包括了地址，联系方法和已知的企业标识；"黄页（Yellow page）"包括了基于标准分类法的行业类别；"绿页（Green Page）"则包括了关于该企业所提供的 Web 服务的技术信息。UDDI 与电话目录非常相似，其形式可能是一些指向文件或是 URL 的指针，而这些文件或 URL 是为服务发现机制服务的。所有的 UDDI 商业注册信息存储在 UDDI 商业注册中心中，一旦 Web Services 注册到 UDDI，客户就可以很方便地查找和定位到所需要的 Web Services。

UDDI 可以实现为公共注册表，以支持全球范围的团体，也可以实现为私有注册表，以支持企业或私人团体。

7.1.4 Web Services 的优势与局限

Web Services 的目标是创建可互操作的分布式应用程序的新平台，在下面几种场合使用 Web Services 将会体现极大的优势。

1. 跨防火墙通信

传统的 Web 应用程序拥有成千上万的用户，而且分布在世界各地，此时客户端和服务器之间的通信将是一个非常棘手的问题，因为客户端和服务器之间通常会有防火墙或者代理服务器。在这种情况下，选用 DCOM 就不是那么简单了，而且通常也不便于把客户端程序发布到每一个用户手中。传统的做法是，采用 B/S 结构，选择浏览器作为客户端，写下一堆 JSP 页面，把应用程序的中间层暴露给最终用户，这样开发难度较大，甚至会得到一个很难维护或根本无法维护的应用程序。

如果中间层组件换成 Web Services 的话，就可以从用户界面直接调用中间层组件，从而省掉建立 JSP 页面的步骤。而要调用 Web Services，可以直接使用 Axis 或 CXF 这样的 SOAP 客户端，也可以使用自己开发的 SOAP 客户端，然后把它和应用程序连接起来，这样不仅缩短了开发周期，减少了代码复杂度，还能够增强应用程序的可维护性。

从实际经验来看，在一个用户界面和中间层有较多交互的应用程序中，使用 Web Services 结构，可以在用户界面编程上节省 20%左右的开发时间。另外，这样一个由 Web Services 组成的中间层，完全可以在应用程序集成等场合下重用，通过 Web Services 把应用程序的逻辑和数据“暴露”出来，还可以让其他客户重用这些应用程序。

2. 应用程序集成

随着企业信息化规模的扩大，经常需要把用不同语言写成的在不同平台上运行的各种应用程序集成起来，而这种集成通常需要花费很大的开发力量。应用程序经常需要从运行在 IBM 主机上的程序中获取数据，或者把数据发送到 UNIX 主机的应用程序中去。即使在同一个平台上，不同软件厂商生产的各种软件也常常需要集成。通过 Web Services，应用程序可以用标准的方法把功能和数据“暴露”出来，供其他应用程序使用。

例如，有一个订单系统，该系统有两大子系统构成：订单录入系统（.NET 实现），用于接收从客户处发来的新订单，包括客户信息、发货地址、数量、价格和付款方式等内容；还有一个订单执行系统（Java 实现），用于实际货物发送的管理，这两个系统运行于不同的平台。一份新订单到来之后，订单录入系统需要通知订单执行程序发送货物。此时可以通过在订单执行系统上面增加一层 Web Services，订单执行程序就可以把执行订单函数“暴露”出来。这样每当有新订单到来时，订单录入系统就可以通过 Web Service 调用这个函数来发送货物了，如图 7-2 所示。

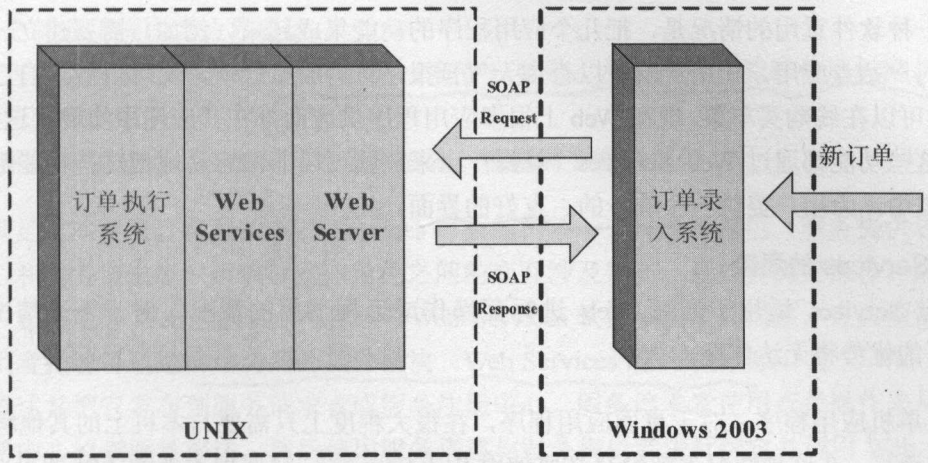


图 7-2　订单系统

3. B2B 集成

Web Services 是 B2B 集成的捷径，通过 Web Services，可以把关键的业务应用“暴露”给指定的供应商和客户。例如，电子订单系统和电子售票系统是常见的 EDI（电子文档交换）应用程序，客户可以在线发送订单，供应商则可以在线发送票务信息。如果通过 Web Services 进行实现，要比 EDI 简单得多，而且 Web Services 运行在 Internet 上，在世界任何地方都可以轻易访问，其运行成本相对来说较低。不过，Web Services 并不像 EDI 那样是文档交换或 B2B 集成的完整解决方案，它只是 B2B 集成的一个高效实现技术，还需要许多其他的部分才能实现集成。

使用 Web Services 来实现 B2B 集成的最大优势是可以轻易实现互操作性，只要把业务逻辑按照 Web Services 规范“暴露”出来，就可以让客户调用这些业务逻辑，而不必考虑它们的系统所运行的平台和开发语言，这样就大大减少了花在 B2B 集成上的时间和成本，让许多原本无法承受 EDI 高昂成本的中小企业也能轻易实现 B2B 集成。

4. 软件重用

软件重用是软件工程的核心概念之一，软件重用的形式很多，重用的程度有大有小，最基本的形式是源代码模块或者类一级的重用，另一种形式是二进制组件重用。像表格、控件或用户自定义控件这样的可重用软件组件，在市场上都占有很大的份额，但是这类软件的重用有一个很大的限制，就是仅限于重用代码，不能重用数据。

Web Services 在重用代码的同时，还可以重用代码背后的数据。使用 Web Services 只需要直接调用远端的 Web Services 就可以了。例如，要在应用程序中确认用户输入的身份证号是否有效，只需把这个地址发送给相应的 Web Services，该 Web Services 就会根据已存的数据信息验证该身份证号的有效性，确认该号码所在的省、市、区等信息，Web Services 的提供商可以按时间或使用次数来对这项服务进行收费。而这样的服务要通过组件重用来实现是很困难的，在这种情况下必须下载完整的包含与身份证号相关信息的数据包，并且需要保证对数据包进行实时更新。

另一种软件重用的情况是，把几个应用程序的功能集成起来。例如，需要建立一个局域网上的门户站点应用，让用户既可以查询天气预报，查看股市行情，又可以管理自己的工作日志，还可以在线购买车票。现在 Web 上很多应用程序供应商都在其应用中实现了这些功能，只要把这些功能都通过 Web Services“暴露”出来，就可以非常容易地把这些功能都集成到门户站点中，为用户提供一个统一的、友好的界面。

5. Web Services 的局限

Web Services 适用于通过 Web 进行互操作或远程调用的情况，对于下述情况，Web Services 的优势将无法体现。

- 单机应用程序。对于桌面应用程序，在很大程度上只需要与本机上的其他程序进行通信，在这种情况下就没有必要使用 Web Services，只要用本地的 API 即可，如常用的 COM 等。
- 局域网应用程序。对于运行于局域网中的程序，一般是由 VC、WinFrom 或 Java 开发而成，其通信往往发生在两个服务器应用程序之间，在这种情况下，使用 DCOM 等技术会比 Web Service 的 SOAP / HTTP 有效得多。

7.2 Web Services 体系结构

Web Services 体系结构是面向对象分析与设计（OOAD）的发展结果，同时也是电子商务解决方案中面向服务体系结构（SOA）、设计、实现与部署所采用的组件化模式的必然结果。它通过采用 SOA 定义一种分布式计算机制，在这种机制下，所有应用程序都被封装为服务并可以通过网络调用。

7.2.1 Web Services 理论模型

Web Services 体系结构的基本原理是基于 SOA 和 Internet 协议的。它代表基于标准和采用这些标准技术的可集成应用程序解决方案。这样就可以确保 Web 服务应用程序的实现方案与标准规范相兼容，从而可以实现与兼容应用程序之间的交互操作。下面是 Web Services 体系结构的一些关键的设计要求。

- 提供通用界面和一致的解决方案模型，将应用程序定义为模块化组件，以便使其成为可发布的服务。
- 使用基于标准的结构模型和协议来定义框架，以便可以在 Internet 上支持基于服务的应用程序。
- 解决各种服务传送场景，包括基于 B2C、B2B、点对点（P2P）和基于企业应用程序集成（EAI）的应用程序通信。
- 使分布式应用程序可成为集中式和分散式应用程序环境，从而支持企业内部和企业

之间应用程序无边界通信。

- 可将服务发布到一个或多个公共（私有）目录，从而使用户能够使用标准机制找到已发布的服务。
- 利用身份验证、授权和其他安全措施，根据需要激活服务。

为满足这些要求，典型的 Web Services 体系结构基于三种逻辑角色（服务提供者、服务注册中心和服务请求者）组成，这些角色之间的交互涉及发布、查找和绑定操作。这些角色和操作一起作用于 Web Services 构件——Web Services 软件模块及其描述。在典型情况下，服务提供者托管可通过网络访问的软件模块（Web Services 的一个实现），定义 Web Services 的服务描述并把它发布到服务请求者或服务注册中心。服务请求者使用查找操作来从本地或服务注册中心检索服务描述，然后使用服务描述与服务提供者进行绑定并调用 Web Services 实现或同它交互。图 7-2 描述了这些操作、提供这些操作的组件及它们之间的交互。

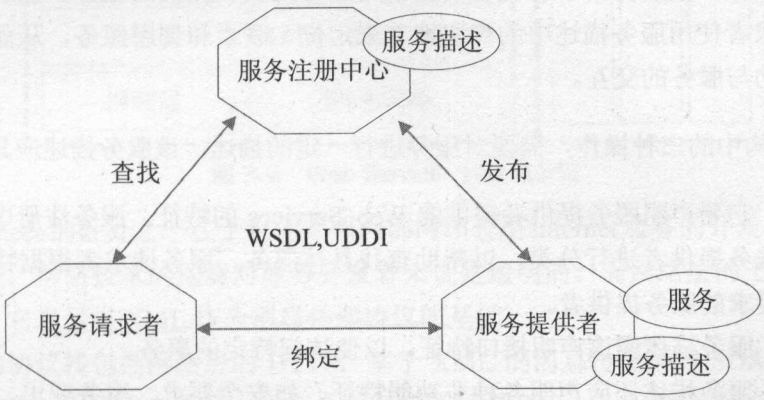


图 7–3　Web Services 体系结构

Web Services 体系结构中共有以下三种角色。

- 服务提供者：发布自己的服务，并且对服务请求进行响应。从企业的角度看，这是服务的所有者；从体系结构的角度看，这是托管访问服务的平台。
- 服务请求者：利用服务注册中心查找所需的服务，然后使用该服务。从企业的角度看，这是要求满足特定功能的商户；从体系结构的角度看，这是寻找并调用服务，或启动与服务的交互的应用程序。服务请求者角色可以由浏览器来担当，由人或无用户界面的程序（例如，另外一个 Web Services）来控制它。
- 服务注册中心：这是可搜索的服务描述注册中心，服务提供者在此发布他们的服务描述，对其进行分类，并提供搜索服务。在静态绑定开发或动态绑定执行期间，服务请求者查找服务并获得服务的绑定信息。对于静态绑定的服务请求者，服务注册中心是体系结构中的可选角色，因为服务提供者可以把描述直接发送给服务请求者。同样，服务请求者可以从服务注册中心以外的其他来源得到服务描述，例如本地文件、FTP 站点、Web 站点、广告和服务发现（Advertisement and Discovery of Services，ADS）或发现 Web Services（Discovery of Web Services，DISCO）。

Web Services 体系结构中的组件必须具有上述一种或多种角色，这些角色之间必须发生以下三个操作：发布服务描述、查询或查找服务描述，以及根据服务描述绑定或调用服务，这些行为可以单次或反复出现。

- 发布操作：使服务提供者可以向服务注册中心注册自己的功能及访问接口，以使服务请求者可以查找它，发布服务描述的位置可以根据应用程序的要求而变化。
- 查找操作：使服务请求者可以通过服务注册中心查找特定种类的服务。在查找操作中，服务请求者直接检索服务描述或在服务注册中心中查询所要求的服务类型。对于服务请求者，可能会在两个不同的生命周期阶段中牵涉查找操作：在设计时为了程序开发而检索服务的接口描述，而在运行时为了调用而检索服务的绑定和位置描述。
- 绑定操作：使服务请求者能够真正使用服务提供者提供的服务。在绑定操作中，服务请求者使用服务描述中的绑定细节来定位、联系和调用服务，从而在运行时调用或启动与服务的交互。

为支持结构中的三种操作，需要对服务进行一定的描述，该服务描述应具有如下特征。

- 首先，它要声明服务提供者提供的 Web Services 的特征。服务注册中心根据某些特征将服务提供者进行分类，以帮助查找具体服务。服务请求者根据特征来匹配那些满足要求的服务提供者。
- 其次，服务描述应该声明接口特征，以便访问特定的服务。
- 最后，服务描述还应声明各种非功能特征，如安全要求、事务要求、使用服务的费用等，接口特征和非功能特征也可以用来帮助服务请求者查找服务。

在 Web Services 中，服务描述和服务实现是分离的，这使得服务请求者可以在服务提供者的一个具体实现正处于开发阶段、部署阶段或完成阶段时，对具体实现进行绑定。另外，Web Services 中的组件之间必须能够进行交互，才能进行上述三种操作。所以，Web Services 体系结构的另一个基本原则就是使用标准的技术，包括服务描述、通信协议以及数据格式等，这样一来，开发者就可以开发出独立于平台、开发语言的 Web Services。

7.2.2 Web Services 协议

Web Services 支持一种标准的协议栈模型，以支持发布、发现和绑定的互操作。图 7-4 展示了一个概念性 Web Services 协议栈模型。

在此模型中，上面的功能层建立在下面功能层提供的功能之上，垂直的功能表示在协议栈的每一层中必须满足的需求。Web Services 协议栈的基础是网络层，Web Services 要被服务请求者调用，就必须通过网络访问。Internet 上可供访问的 Web Services 必须使用普遍部署的网络协议，而 HTTP 凭借其普遍性，成为 Internet 可用的 Web Services 真正的标准网络协议。

Web Services 还可以支持其他 Internet 协议，包括 FTP、SMTP、MQ（消息队列）、IIOP（Internet ORB 间协议）上的远程方法调用（Remote Method Invocation，RMI）、E-mail 等，应使用哪种网络协议和应用程序的具体需求有关。

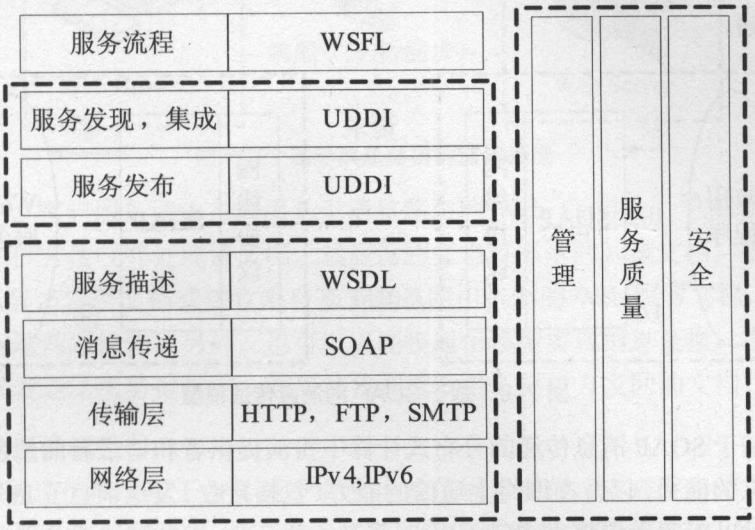


图 7-4　Web Services 协议栈模型

Web Services 的好处之一在于它为专用内部网和公用 Internet 服务的开发和使用提供了统一的编程模型，网络技术的选择对服务开发者来说是透明的。而网络层的上面是基于 XML 的消息传递，它表示将 XML 作为消息传递协议的基础。

最简单的协议栈包括网络层的 HTTP，基于 XML 的消息传递层的 SOAP 协议以及服务描述层的 WSDL，如图 7-5 所示。所有企业间或公用 Web Services 都应该支持这种可互操作的基础协议栈，特别是企业内部或专用 Web Services，也应支持其他的网络协议和分布式计算技术。

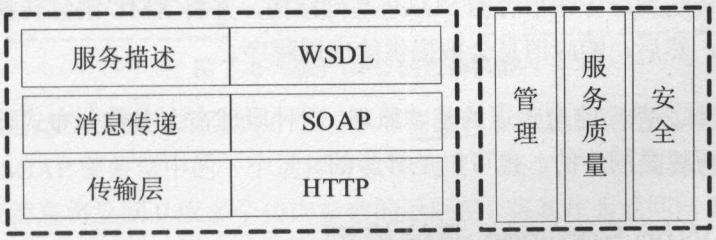


图 7-5　Web Services 基础协议栈

Web Services 体系结构的基础是 XML 消息传递，XML 消息传递的行业标准是 SOAP。SOAP 是一种简单的、轻量级的基于 XML 的机制，用于在网络应用程序之间进行结构化数据交换，它包括以下三个部分。

- 一个定义描述消息内容的框架的信封。
- 一组表示应用程序定义的数据类型实例的编码规则。

■ 表示远程过程调用和响应的约定。

SOAP 可以和各种网络协议（如 HTTP、SMTP、FTP 等）结合使用，或者用这些协议重新封装后使用。图 7-6 展示了基于 SOAP 的 XML 消息传递和网络协议如何组成 Web Services 体系结构的基础。

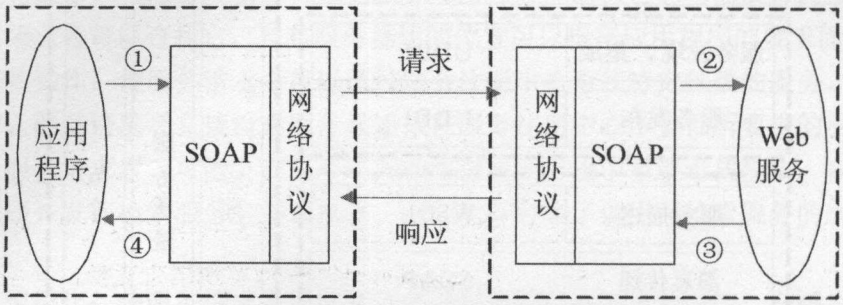


图 7-6 基于 SOAP 的消息传递模型

网络层在基于 SOAP 消息传递的分布式计算中扮演提供者和请求者的角色，它具有构建、解析 SOAP 消息的能力，以及在网络上通信的能力（包括接收、发送消息）。应用程序与 SOAP 的集成通常通过以下四个步骤来实现。

01 服务请求者（应用程序）创建一条 SOAP 消息，然后将此信息和服务提供者的网址一起提供给 SOAP 基础结构，SOAP 基础结构与一个底层网络协议交互，从而在网络上将 SOAP 消息发送出去。

02 网络基础结构在将消息传送到服务提供者（Web Services 服务器）的 SOAP 基础结构，SOAP 基础结构负责将 XML 消息转换为特定的消息对象。

03 Web Services 服务器负责处理请求信息并生成一个响应（也是 SOAP 消息），响应的 SOAP 消息被提供给 SOAP 基础结构，其目的地是网络上的服务请求者。

04 响应消息传送到服务请求者的 SOAP 基础结构，并将 XML 消息转换为目标编程语言中的对象，然后，响应消息会被提供给应用程序。

上述步骤描述了请求/响应传送的基本原理，这种原理在大多数分布式计算环境中都很常见。该传送过程可以是同步的，也可以是异步的。

7.2.3 Web Services 通信模型

在 Web Services 体系结构中，根据功能要求的不同，可以实现基于消息路由的同步/异步通信模型或基于 RPC 的异步模型。

1. 基于消息路由的通信模型

基于消息路由的通信模型定义松散关联和文档驱动的通信。调用基于消息交换的服务，服务请求方无需等待响应。图 7-7 描述了 Web Services 体系结构中基于消息交换的通信模型。

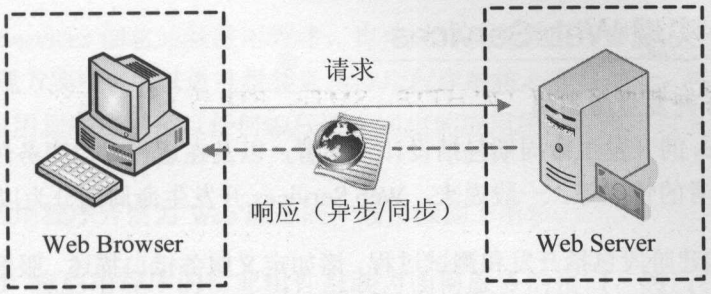


图 7-7　基于消息路由的通信模型

在图 7-7 中，客户服务请求方调用基于消息路由的 Web Services；它通常发送一个完整的文档，服务提供方接收并处理该文档，然后返回（也可不返回）该文档。根据实现方式的不同，客户可以异步发送文档或接收来自基于消息路由机制的 Web 服务文档，但不能在一个实例中同时执行这两项操作。另外，也可以以同步通信模型实现消息交换，在这种模型下，客户将服务请求发送给服务提供方，然后等待和接收服务提供方发回的文档。

2. 基于 RPC 的通信模型

基于 RPC 的通信模型用于定义基于请求/响应的同步通信。客户发出请求后，继续任何操作之前将等待服务器发回响应。尤其对于实现 CORBA 或 RMI 通信来说，基于 RPC 的 Web 服务是紧密关联的，并且经由到客户应用程序的远程对象实现的。图 7-8 描述了 Web Services 体系结构中基于 RPC 的通信模型。

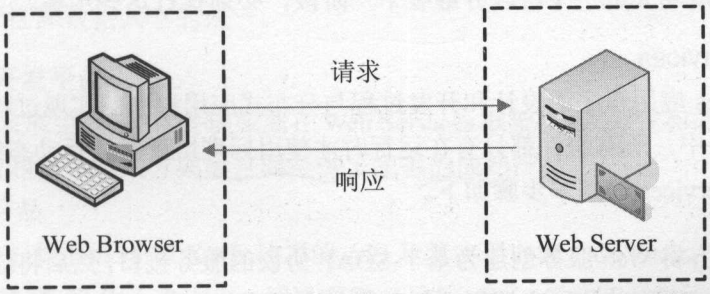


图 7-8　基于 RPC 的通信模型

根据 RPC 约定，SOAP 消息用包含 0 个或多个参数和返回值的方法名称表示。每个 SOAP 请求消息表示对 SOAP 服务器中的一个远程对象的调用，每个方法调用都有 0 或多个参数。同样，SOAP 响应消息将返回 0 或多个传出参数的返回值，将其作为返回结果。在 SOAP RPC 请求和响应中，方法调用被串行化为 SOAP 编码规则定义的、基于 XML 的数据类型。另外，使用基于 RPC 的通信，服务提供方和请求方都可以分别注册和发现服务。

采用何种通信模型（消息路由和 RPC）也取决于 Web 服务结构及其采用的协议。SOAP1.2 同时支持这两种通信模型。

7.2.4 实现 Web Services

1. Web Services 开发生命周期

Web Services 的开发生命周期包括设计、部署，以及在运行时对服务注册中心、服务提供者和服务请求者的管理等。一般说来，Web Services 开发生命周期分为以下四个阶段。

- 构建。构建阶段包括开发和测试过程，诸如定义服务接口描述、服务实现和定义 Web Services 实现描述等都在此阶段完成。此过程可以通过创建新的 Web Services、把现有的应用程序封装成 Web Services 以及将其他 Web Services 或应用程序重组成新的 Web Services 来实现。
- 部署。此阶段包括向服务注册中心发布服务接口和服务实现定义，以及把 Web Services 的可执行文件部署到执行环境中。
- 运行。Web Services 部署完毕后，服务请求者可以通过服务注册中心进行特定 Web Services 的查找，并根据规范实现绑定、调用操作。
- 管理。管理阶段包括持续的管理和经营 Web Services 应用程序。如解决安全性、可用性、性能、服务质量、业务流程等问题。

创建 Web Services 的过程与创建其他类型应用程序的过程别无二致，事实上就是一个“设计与实现 → 部署与发布 → 发现与调用”的过程。这个生命周期内的每个阶段都有一些特有的步骤，要想成功完成一个阶段并继续下一阶段，必须执行这些步骤。

2. 实现 Web Services

Web Services 应用程序的设计和开发过程与分布式应用程序的实现过程没有不同之处。只是在 Web 服务中，所有组件都只有在运行时才使用标准协议时进行动态绑定。

实现 Web Services 的基本步骤如下。

01 服务提供方将 Web 服务创建为基于 SOAP 协议的服务接口，然后将这些服务部署到服务容器中，以便其他用户调用。服务提供方同时将这些 Web 服务创建基于 WSDL 的服务描述，这种描述使用统一的方法来标识服务位置、操作及其通信模式，以定义客户端和服务容器。

02 服务提供方使用服务代理注册基于 WSDL 的服务描述，服务代理方通常是 UDDI 注册表。

03 UDDI 注册表将服务描述存储为绑定模板和到服务提供方环境中 WSDL 的 URL。

04 服务请求方通过查询 UDDI 注册表找到所需服务并获取绑定信息和 URL，以确定服务提供方。

05 服务请求方使用绑定信息激活服务提供器，并检索已注册服务的 WSDL 服务描述，通过创建客户代理应用程序，建立与服务器间的通信。

06 最后，服务请求方与服务提供方通信，并通过调用服务容器中的服务进行信息交换。

可以将 Web Services 创建为新应用程序，也可以将现有应用程序更改为 Web Services。在 Web 服务的实现方案中，可以通过封装底层应用程序的核心业务概念，将现有应用程序发布为服务。这些应用程序可以是以任何编程语言实现的应用程序，也可以是在任何平台上运行的应用程序。

将现有商业应用程序开发为 Web 服务的一般步骤如下所示。

01 使用 SOAP 将应用程序的商业组件封装为面向服务的接口，然后将其部署在 Web Services 服务容器中，发布为 Web 服务。基于这些接口，服务容器可以处理所有传入的、基于 SOAP 的消息交换操作，并将其映射为底层商业应用程序的方法或参数。

02 生成基于 WSDL 的服务描述，这些服务描述将驻留在服务容器中。这些基于 WSDL 的服务描述将在 UDDI 注册表中发布为服务模板及其位置 URL。

03 服务请求方通过查询 UDDI 注册表找到所需服务并获取绑定信息和 URL，然后连接服务提供方，以获取 WSDL。

04 要调用服务提供方发布的服务，服务请求方（服务发送环境）需要根据 WSDL 中定义的服务描述实现基于 SOAP 的客户接口。

上述步骤通常适用于所有级别的 Web Services 开发，而不必考虑目标应用程序的环境（如 J2EE、CORBA、Microsoft.NET）或应用程序本身（如 Java、C++、VB 等）以及原有应用程序和主架构环境。因此，实现 Web 服务实际上是 J2EE、CORBA、.NET 和其他基于 XML 的应用程序与互操作性和数据共享的统一。

3. Web Services 开发平台

Web Services 发展至今，各软件供应商在 Web Services 框架核心、开发和部署 Web Services 的体系结构、平台和软件解决方案等方面提供了完善的成熟产品。

（1）Sun 的产品

Sun 公司发布了基于 Java 和 XML 技术的 API 及其实现方案 JAX Pack，用于开发、测试 Java 和基于开放式标准的 Web Services 解决方案。另外，Sun 公司还发布了一组全面的、专用于 Web Services 的开发工具包 JWSDP（Java Web Services Developer Pack）。

JWSDP 提供了库和工具的集合，包含了开发和测试 Web 服务的必需组件。JWSDP 支持行业标准，这就能够确保其与标准组织所发布的技术和规范具有互操作性，例如万维网协会（W3C）以及结构标准推动组织（OASIS）等。JWSDP 除了提供标准的接口库外，还提供了每个库的引用实现，同时，JWSDP 还提供多种辅助工具，例如 WSDLstub 编译器，它能够生成一个 WSDL 文件，用于 Web 服务和独立的 Web 服务 UDDI2.0 注册表。

JWSDP 包括下列常用组件。

- JAXP：XML 处理的 Java API，提供了解析和转换 XML 文档的标准化接口。
- JAXM：JAXM（Java API for XML Messaging）提供 SOAP1.1 带附件的 SOAP 标准接口，以便 Java 程序员能够方便地发送和接收 SOAP 消息。JAXM 给厂商提供了一

种机制，既可以用于支持可靠的消息传输，也可以用于部分填充特定的基于 SOAP 协议（如 ebXML）的 SOAP 消息。

- JAX-RPC：基于 XML 远程过程调用的 Java API（Java API for XML-based Remote Procedure Calls，JAX-RPC）提供了一种机制，使得可以通过基于 SOAP 的消息跨网络调用对象。JAX-RPC 的作用与 RMI 大致相同，都是通过创建的插件对象调用远程对象，从概念上讲，可以同样地使用这两个系统。JAX-RPC 和 RMI 的不同之处在于两台机器之间所传递的数据格式。RMI 使用底层的 Java 特有的协议，而 JAX-RPC 使用的则是 XML。
- JAXR：JAXR（XML 注册中心的 Java API，Java API for XML Registries）提供了查询注册中心的抽象接口；JAXR 可用来让用户不必了解 UDDI 或 ebXML RegRep 的细节。
- JSSE：JSSE（Java Secure Socket Extension）提供了一种机制，可用于在加密的网络连接上进行通信，并且管理与加密有关的密钥。JSSE 提供了免版税的 SSL v3（安全套接字层，Secure Sockets Layer）的实现和 TLS（传输层安全，Transport Layer Security）1.0 的支持。JSSE 还提供了附加的 URL 处理器，以便 java.net.URL 类能够理解和处理 HTTP URL。
- JSTL：JSP 的标准标签库。
- ANT 和 TOMCAT：ANT 是一个开放工源代码构建工具。Tomcat 是 JSP 和 Servlet 标准的引用实现。它同样允许开发基于 Servlet 的 Web 服务托管。
- WSDP：WSDP（WEB 服务开发人员包，Web Services Developer Package）Registry 服务器。这是一个简单的 UDDI 服务器，可用于开发和测试。

Sun ONE Application Serve 产品套装是 Sun 公司推出的基于 Java/J2EE 的用于构建和部署 Web Services 的解决方案。Sun ONE 套装的体系结构基于 SOAP、WSDL 和 UDDI 等开放式标准，提供了一种可供用户和开发人员迁移到下一代 Web 服务的全面的 Web 服务软件环境。

Sun ONE 主要的优点是不会出现任何客户锁定问题或其他所有权解决方案导致的问题。它的最主要的缺点是对 Web Services 的描述和发现的两个标准（WSDL 和 UDDI）的支持尚不完善。

（2）IBM Web Services

IBM 电子商务是 IBM 公司基于理念性体系结构和开放式标准的产品，用于提供 Web 服务的开发和部署。IBM 提供的产品基于 Java/J2EE、SOAP、WSDL 和 UDDI 等 Web 服务标准，全面反映了 Web 服务技术在动态电子商务中的应用。

IBM WebSphere Application Server 提供用于部署基于 Web 服务的应用程序的基础结构解决方案。IBM 还为开发人员提供 Web 服务工具包（Web Services ToolKit，简写 WSTK），用于创建、发布和测试基于 XML、SOAP、WSDL 和 UDDI 等开放标准的 Web 服务解决方案的运行环境，使用 WSTK 不必对现有应用程序进行重编程即可生成 WSDL 包装器。

（3）Microsoft.NET

Microsoft.NET 提供了用于开发、部署基于标准的 Web 服务和所有类型的应用程序的.NET 平台的框架和编程模型。这一框架定义了三个层：Microsoft 操作系统、企业服务器和使用 Visual Studio 的.NET 构件块。基于.NET 的 Web 服务界面使用支持 SOAP、WSDL 和 UDDI 等标准的.NET 构件开发实现。

Microsoft 提出的 Web Services 解决方案及开发工具无疑是最便于使用的，但其缺点也很明显——无法在 Windows 之外的平台上使用。

小结

通过本章的学习，学生应该能够学会：

- Web Services 是建立于 SOA 基础之上的最新分布式计算技术。
- Web Services 建立在 XML 标准上，可以使用任何编程语言、协议或平台开发。
- 用于构建和使用 Web 服务主要有四种标准和技术：XML、SOAP、WSDL 和 UDDI。
- SOAP 是一种基于 XML 的轻量级消息交换协议。
- WSDL 是一种 XML 格式，用于描述网络服务及其访问信息。
- UDDI 提供了在 Web 上描述并发现商业服务的框架。
- Web Services 应用在跨防火墙通信、应用程序集成、B2B 集成和数据重用等场合中会体现极大的优势。
- 在单机应用程序和局域网应用程序中，Web Services 无法体现其优势。
- Web Services 体系结构基于三种逻辑角色——服务提供者、服务注册中心和服务请求者。
- Web Services 体系结构中，角色之间可以单次或反复出现发布、查找、绑定三种操作。
- 最简单的协议栈包括网络层的 HTTP，基于 XML 的消息传递层的 SOAP 协议以及服务描述层的 WSDL。
- 在 Web Services 体系结构中，根据功能要求的不同，可以实现基于消息路由的同步/异步通信模型或基于 RPC 的异步模型。

Web Services 开发生命周期分为构建、部署、运行、管理共四个阶段。

练习

1. Web Services 应用程序具备如下哪些特征_____。（多选）
   A. 封装性
   B. 松散耦合
   C. 使用标准协议规范

D. 高度可集成性

2. Web Services 应用的优势体现在如下哪种场景中_____。（多选）

A. 跨防火墙通信

B. 应用程序集成

C. B2B 集成

D. 数据重用

3. Web Services 体系结构基于哪三种逻辑角色_____。（多选）

A. 服务提供者

B. 服务注册中心

C. 服务请求者

D. 消息

4. 用于构建和使用 Web 服务主要有四种标准和技术：_____、_____、_____和_____。

5. Web Services 体系结构中，角色之间可以单次或反复出现三种操作_____、_____和_____。

6. 最简单的协议栈包括_____、_____及_____。

7. 简要描述 Web Services 应用程序的特征。

8. 简要描述 Web Services 的主要技术及各种技术的作用。

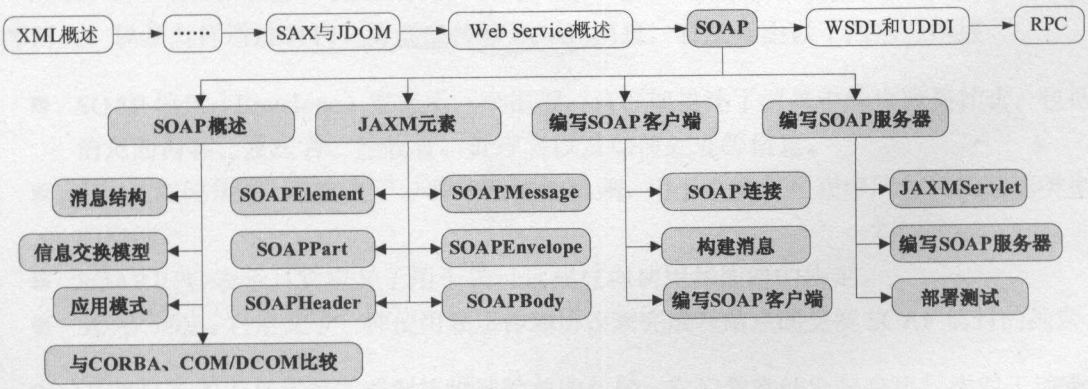
9. 什么是 SOAP？简要描述 SOAP 的组成。

第 8 章　SOAP

本章目标

- 了解 SOAP 的应用背景
- 掌握 SOAP 的消息结构
- 掌握 SOAP 消息交换模型
- 了解 SOAP 的常用应用模式
- 了解 SOAP 与 COM、CORBA 的区别
- 掌握常用的 JAXM 元素
- 了解 SOAP 连接的种类
- 掌握 SOAP 客户端的实现方式
- 掌握 SOAP 服务端的实现方式

学习导航



任务描述

【描述 8.D.1】

基于消息发送者到接收者的点对点直接连接，创建客户端，向服务器发送 SOAP 消息，并处理服务器的返回消息。

【描述 8.D.2】

基于消息发送者到接收者的点对点直接连接，创建 SOAP 服务端，接收、处理 SOAP 消息（将 SOAP 消息存入指定文件），并返回 SOAP 响应消息。

8.1　SOAP 概述

SOAP 是一种基于 XML 的消息规范，其描述了数据消息的格式以及一整套串行化规则，包括结构化类型和数组。SOAP 可以使用各种 Internet 标准协议（如 HTTP 和 SMTP）作为其消息传输工具，还可以提供表示 RPC 和文档驱动的消息交换等通信模型的约定。这样就可以在分布式环境中实现应用程序之间的通信，并可以实现网络上异构应用程序之间的交互操作。SOAP 已经受到主要的 IT 供应商和 Web 开发人员的广泛欢迎，从而在基于 Web Service 的商业应用程序中流行起来，被广为采用。

8.1.1　SOAP 介绍

SOAP（Simple Object Access Protocol，简单对象访问协议）是一个基于 XML 的协议，它是分布式系统之间交换信息的轻量级方法，它广泛支持各种系统和 RPC 以及 Messaging 等通信模型，是一种非常好的跨平台和跨厂商的技术，并且得到了各大软件供应商的支持。目前最新版本的 SOAP 1.2 是 Sun Microsystems、IBM、BEA、Microsoft 和 Oracle 等供应商领导的 W3C XML 工作组推出并维护的。

SOAP 的两个目标是简单性和可扩展性，这就意味着有一些传统消息系统或分布式对象系统中的某些性质将不是 SOAP 规范的一部分。SOAP 是一种分布式计算中的技术，它被有意地设计为轻量级的协议，它的某些功能是和其他一些机制（如 RMI）共有的，但也存在许多不同的功能，如不支持通过引用传递对象、对象激活、消息批处理等。SOAP 也被设计成可扩展的，而简单性和可扩展性意味着 SOAP 的使用不包含任何特定的编程模式，给定实现的语义是极为灵活的。

SOAP 基于 XML 语言和 XSD 标准，其定义了一套编码规则，该规则定义如何将数据表示为消息，以及怎样通过 HTTP 协议来传输 SOAP 消息，它主要由以下四部分组成。

- SOAP 信封（Envelope）定义了一个框架，该框架描述了消息中的内容是什么，包括消息的内容、发送者、接收者、处理者以及如何处理等信息。
- SOAP 编码规则，它定义了一种序列化的机制，用于交换应用程序所定义的数据类型的实例。
- SOAP RPC 表示，它定义了用于表示远程过程调用和应答的协定。
- SOAP 绑定，它定义了一种使用底层传输协议来完成在结点间交换 SOAP 信封的约定。

SOAP 消息基本上是从发送端到接收端的单向传输，它们常常结合起来执行类似于请求/应答的模式。不需要把 SOAP 消息绑定到特定的协议，SOAP 可以运行在任何其他传输协议（HTTP、SMTP、FTP 等）上。另外，SOAP 提供了标准的 RPC 方法来调用 Web Services，以请求/响应模型运行。

注意　完整地介绍 SOAP 超出了本书的范围，关于 SOAP 的详细介绍，读者可参考 http://www.w3.org/TR/soap/。

8.1.2　SOAP 消息结构

SOAP 定义了基于 XML 的消息规则和机制，可用于实现应用程序之间的通信，更重要的是，SOAP 采用了 XML Schema 和命名空间等 XML 语法和标准作为其消息结构的组成部分。所有的 SOAP 消息都使用 XML 编码，一条 SOAP 消息就是一个普通的 XML 文档，该文档包含下列元素。

- Envelope（信封）元素，必选，可把此 XML 文档标识为一条 SOAP 消息。
- Header（报头）元素，可选，包含头部信息。
- Body（主体）元素，必选，包含所有的调用和响应信息。
- Fault 元素，位于 Body 内，可选，提供有关处理此消息所发生错误的信息。
- attachment（附件）元素，可选，可通过添加一个或多个附件扩展 SOAP 消息。

图 8-1 描述了带附件的 SOAP 消息的基本结构。

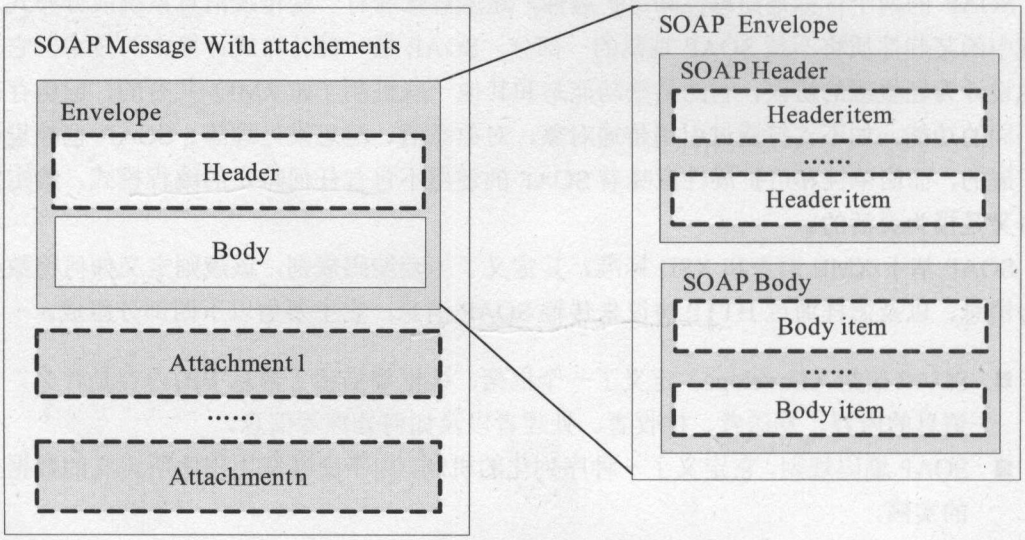


图 8–1　带附件的 SOAP 消息的结构

下述示例通过代码描述了 SOAP1.1 规范下的消息的基本结构。

```
<?xml version="1.0"?>
<soap-env:Envelope xmlns:soap-env="http://schemas.xmlsoap.org/soap/envelope/"
soap-env:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
    <soap-env:Header>
        <!--报头定义-->
    </soap-env:Header>
```

```
    <soap-env:Body>
        <!--消息体定义-->
        <soap-env:Fault>
            <!--相关错误处理信息-->
        </soap-env:Fault>
    </soap-env:Body>
</soap-env:Envelope>
```

SOAP 虽然是 XML 文档，但其编写需要满足如下语法规则。

- SOAP 消息必须用 XML 来编码。
- SOAP 消息必须使用 SOAP Envelope 命名空间。
- SOAP 消息必须使用 SOAP Encoding 命名空间。
- SOAP 消息不能包含 DTD 引用。
- SOAP 消息不能包含 XML 处理指令。

1. SOAP Envelope

SOAP Envelope（信封）是 SOAP 消息结构的主要容器，也是 SOAP 消息的根元素，它必须出现在每个 SOAP 消息中，用于把此 XML 文档标识为一条 SOAP 消息。在 SOAP 中，使用 XML 命名空间将 SOAP 标识符与应用程序的特定标识符区分开，将 SOAP 消息的元素的作用域限制在一个特定的领域。下述代码描述了 SOAP 的一个空信封。

```
<soap-env:Envelope xmlns:soap-env="http://schemas.xmlsoap.org/soap/envelope/"
soap-env:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
</soap-env:Envelope>
```

上述代码中的 SOAP 信封不包含主体（Body）元素，所以不是一个有效的 SOAP 消息，但是它说明了 SOAP 消息中有关命名空间的重要一点：SOAP 命名空间由该信封定义为具有“http://schemas.xmlsoap.org/soap/envelope/”的 URI 和“soap-env”前缀。这是因为按照 SOAP1.1 规范，SOAP 消息中的所有元素必须由这个特定的命名空间进行限定，如果使用了不同的命名空间，应用程序会发生错误，并抛弃此消息。SOAP 的 encodingStyle 属性用于定义在文档中使用的数据类型。此属性可出现在任何 SOAP 元素中，并会被应用到元素的内容及元素的所有子元素上。

对于 SOAP1.2 规范，相关联的命名空间是“http://www.w3.org/2001/12/soap-envelope”，对应的空信封代码如下所示。

```
<soap:Envelope xmlns:soap="http://www.w3.org/2001/12/soap-envelope"
 soap:encodingStyle="http://www.w3.org/2001/12/soap-encoding">
</soap:Envelope>
```

注意 如无特殊说明，本章关于 SOAP 的介绍都是基于 SOAP1.1，关于 SOAP1.2 的详细介绍可参考 http://www.w3.org/TR/soap12。

SOAP Envelope 的语法规则介绍如下。

- 元素名为 Envelope，该元素必须在 SOAP 消息中作为根元素出现。
- 该元素可以包含命名空间声明和额外的属性，如果声明额外的属性，必须使用命名空间修饰。
- Envelope 可以包含额外的子元素，但必须使用命名空间修饰并且跟在 SOAP Body 元素之后。

2. SOAP Header

SOAP Header（报头）元素应当作为 SOAP Envelope 的第一个直接子元素，它必须使用有效的命名空间。Header 还可以包含 0 个或多个可选的子元素，这些子元素称为 Header 项，所有 Header 项都必须是完整修饰的，即必须由一个命名空间 URI 和局部名组成，不允许没有命名空间修饰的 Header 项存在。

Header 元素用于与消息一起传输附加消息，如身份验证或事务信息。Header 元素也可以包含某些属性。SOAP 在默认的命名空间中定义了三个属性：actor、mustUnderstand 以及 encodingStyle。这些被定义在 SOAP 头部的属性可通知容器如何对 SOAP 消息进行处理。

- encodingStyle 属性用于指明 Header 项的编码风格。
- mustUnderstand 属性用于指明消息的接收方对 SOAP 报头的处理是否是必须的。如果设置为 1，指示处理此头部的接收者必须认可此元素，假如此接收者无法认可此元素，则在处理此头部时必须失效。默认值为 0。
- actor 属性可被用于将 Header 元素寻址到一个特定的接收者。

下面的 SOAP 信封中包含一个可忽略身份验证的消息报头。

```
<soap-env:Envelope xmlns:soap-env="http://schemas.xmlsoap.org/soap/envelope/">
    <soap-env:Header>
        <auth:UserID xmlns:auth="some-URI">
            Admin
        </auth:UserID>
    </soap-env:Header>
</soap-env:Envelope>
```

在该消息报头中，使用前缀“auth”和 URI “some-URI”定义了新命名空间的附加元素。这样的命名空间可以由任何人进行定义，本例中的实际元素名（UserID）表明它包含身份验证信息。如果消息的接收者理解 UserID 元素的意思（在已定义的命名空间内），它应该处理该元素（如，承认 Admin 是一个经过身份验证的账户名）。如果消息的接收者不知道如何处理该元素，则它可以忽略。

3. SOAP Body

SOAP Body（主体）元素作为子元素包含在 SOAP 信封中。SOAP1.1 规范规定一条消息

中必须有一个或多个 SOAP Body，SOAP 消息的 Body 块可以包含以下任何元素。

- RPC 方法及其参数。
- 目标应用程序（消息接收者）专用数据。
- 报告故障和状态信息的 SOAP 故障（Fault）。

在 SOAP 消息中，所有的 Body 元素必须是 SOAP Envelope 元素的直接子元素。若该消息中包含 Header 元素，则 Body 元素必须直接跟随 Header 作为兄弟元素；若 Header 不出现，则它必须是 Envelope 的第一个直接子元素。

所有 Body 元素的直接子元素都称为 Body 项，Body 项都必须是 Body 元素的直接子元素，Body 项必须由命名空间修饰。Body 项自身可以包含下级子元素，但这些元素不是 Body 项，而是 Body 项的内容。

下述代码显示一个消息主体，该主体表示用于从“www.fruit.com”获取苹果（Apples）的价格信息的 RPC 调用。

```
<soap-env:Envelope xmlns:soap-env="http://schemas.xmlsoap.org/soap/envelope/">
    <soap-env:Body>
        <m:GetPrice xmlns:m="http://www.fruit.com/prices">
            <m:Item>Apples</m:Item>
        </m:GetPrice>
    </soap-env:Body>
</soap-env:Envelope>
```

请注意，上面的 m:GetPrice 和 Item 元素使用了“http://www.fruit.com/prices”命名空间，它们是应用程序专用的元素，它们并不是 SOAP 标准的一部分。

对于上述请求，SOAP 响应可能如下：

```
<soap-env:Envelope xmlns:soap-env="http://schemas.xmlsoap.org/soap/envelope/">
    <soap-env:Body>
        <m:GetPriceResponse xmlns:m="http://www.fruit.com/prices">
            <m:Price>1.90</m:Price>
        </m:GetPriceResponse>
    </soap-env:Body>
</soap-env:Envelope>
```

4. SOAP Fault

SOAP Fault（错误）元素用于在 SOAP 消息中传输错误及状态信息。如果 SOAP 消息需要包括 SOAP Fault 元素，它必须作为一个 Body 项出现，而且至多出现一次。SOAP Fault 元素有以下几个子元素。

- faultcode：必须在 SOAP Fault 元素中出现，该元素用来标识故障，它包含用于标识该 SOAP 应用程序的故障或状态的标准值。这些 faultcode 值的命名空间标识符在

http://schemas.xmlsoap.org/soap/envelope/中定义。

- faultstring：该元素提供 SOAP 故障的可读描述。
- faultactor：该元素描述在消息路径中错误的引发者。类似于 SOAP actor 属性，不过它记录的不是 Header 的接收者，而是指示错误源。
- detail：该元素提供与已定义的 Body 块相关的，特定于应用程序的故障或状态信息。detail 元素的所有直接子元素都称为 detail 项。

下述代码演示了如何在 SOAP 消息中表示 SOAP Fault。

```
<soap-env:Envelope xmlns:soap-env="http://schemas.xmlsoap.org/soap/envelope/">
    <soap-env:Body>
        <soap-env:Fault>
            <faultcode>soap-env:MustUnderstand</faultcode>
            <faultstring>Header element missing</faultstring>
            <faultactor>http://www.fruit.com/prices</faultactor>
            <detail>
                <fruit:error xmlns:fruit="http://www.fruit.com/prices">
                    <problem>The Furit name missing.</problem>
                </fruit:error>
            </detail>
        </soap-env:Fault>
    </soap-env:Body>
</soap-env:Envelope>
```

下述代码显示了因服务器故障引起的 SOAP Fault。

```
<soap-env:Fault>
    <faultcode>soap-env:Server</faultcode>
    <faultstring>Server Internal failure</faultstring>
    <faultactor>http://www.fruit.com/prices</faultactor>
</soap-env:Fault>
```

5. 附件

按照SOAP1.1规范的规定，SOAP消息可以包含XML格式的主SOAP信封，以及含ASCII或二进制等任何数据格式的 SOAP 附件。如果 SOAP 消息包含附件，那么 SOAP 消息将是一个 MIME 编码的消息，它包含 SOAP 内容和一个或多个其他类型的附件。因此 SOAP 消息实际上分为以下两种类别。

- 仅包含 XML 内容的消息。
- MIME 编码的消息，包含初始的 XML 有效内容以及任何数量的附件。这些附件可以是任何类型的数据。

注意　多用途 Internet 邮件扩展（Multi-purpose Internet Mail Extensions，MIME）是一组技术规范，其目的是使用不同字符集来传递文本，也可以在计算机之间传递各种各样的多媒体数据，关于 MIME 的详细介绍，读者可参考相关资料。

SOAP 消息结构中的每一个 MIME 附件都使用 Content-ID 或 Content-Location 作为附件的标签引用。SOAP 消息头和消息正文也可以指向消息中的这些标签。每个消息附件都可使用 Content-ID（通常使用 URL 模式的 href 属性）或 Content-Location（与附件关联的 URI 引用）标识。下述代码使用 apples.gif 作为附件，说明 Content-ID 引用在 SOAP 消息中的应用。

```
MIME-Version:1.0
Content-Type:Multipart/Related;boundary=MIME_boundary;type=text/xml;
start="<http://www.fruit.com/coverdetail.xml>"
Content-Description:SOAP message description
--MIME_boundary--
Content-Type: text/xml;charset=UTF-8
Content-Transfer-Encoding:8bit
Content-ID:<http://www.fruit.com/coverdetail.xml>
Content-Location: http://www.fruit.com/coverdetail.xml
<soap-env:Envelope xmlns:soap-env="http://schemas.xmlsoap.org/soap/envelope/">
    <soap-env:Body>
        <!--SOAP BODY-->
        <theCoverPage href="http://www.fruit.com/img/apples.gif"/>
        <!--SOAPBODY-->
    </soap-env:Body>
</soap-env:Envelope>
--MIME_boundary--
Content-Type:image/gif
Content-Transfer-Encoding:binary
Content-ID:<http://www.fruit.com/img/apples.gif>
Content-Location:http://www.fruit.com/img/apples.gif
<!-- binary GIF image -->
--MIME_boundary--
```

该消息中使用了绝对 URI 引用项，以用于 Content-Location 消息头。

注意　有关 SOAP 附件的详细介绍，读者可参考 http://www.w3.org/TR/SOAP-attachments。

8.1.3　SOAP 消息交换模型

SOAP 消息交换是从发送方到接收方的一种传输方法。从本质上说，SOAP 是一种无态（stateless）协议，它提供复合的单向消息交换框架，以便在称之为 SOAP 结点的 SOAP 应用程序之间传输 XML。

1. SOAP 结点

SOAP 结点表示 SOAP 消息路径的逻辑实体，用于执行消息路由或处理。SOAP 结点既可以是 SOAP 消息的发送者，也可以是 SOAP 消息的接收方，还可以是 SOAP 消息发送方和接收方间的 SOAP 消息中介。

在 SOAP 消息交换模型中，中间方为一种 SOAP 结点，负责提供发送消息的应用程序和接收消息的应用程序之间的消息交换和协议路由功能。中间方结点驻留在发送结点和接收结点之间，负责处理 SOAP 消息头中定义的部分消息。SOAP 发送方和接收方之间可以有 0 或多个 SOAP 中间方，它为 SOAP 接收方提供分布式处理机制，如图 8-2 所示。

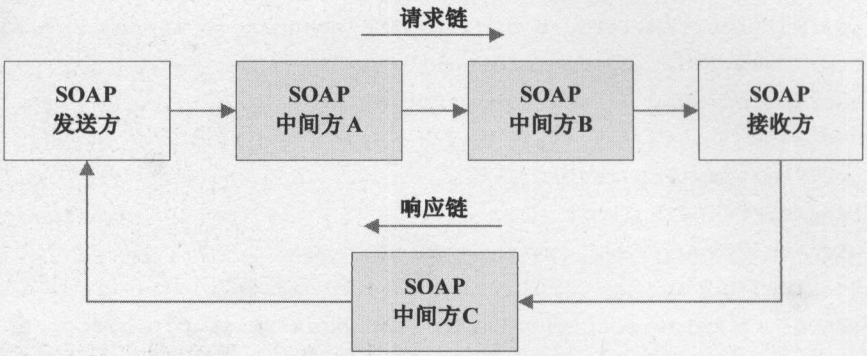


图 8-2　SOAP 消息交换模型

图 8-2 描述了发送方、接收方及其中间方齐全的完整消息交换模型示例。该示例中，消息源于发送方，经中间方 A 和 B 发送到接收方，从而形成了一条请求链；然后，响应消息从发送方（请求链的接收方）经中间方 C 发送回接收方（请求链的发送方），从而形成响应链。

一般说来，SOAP 消息中间方分为以下两种。

- 转发中间方：这一类型的中间方通过在所转发消息的 SOAP 消息头块中描述和构造语义和规则，从而实现消息处理。
- 活动中间方：这一类型的中间方利用一组功能为接收方结点修改外部绑定消息，从而提供更多的处理操作。

在 SOAP 消息交换路径中，借助于 SOAP 中间方，使得分布式处理模型在 SOAP 消息交换中得以实现。通过使用 SOAP 中间方，可以向 SOAP 应用程序中集成各种功能（如转发、过滤、事务、安全、日志记录、智能路由等）。

2. SOAP actor 属性

一个 SOAP 消息从发送方到接收方的过程中，可能沿着消息路径经过一系列 SOAP 中间结点。SOAP 中间结点是可以接收、转发 SOAP 消息的应用程序。中间结点和接收方由 URI 区分。可能 SOAP 的接收方并不需要消息的所有部分，而在消息路径上的一个和某几个中间结点可能需要这些内容。头元素的接收者扮演的角色类似于一个过滤器，防止这些只发给本接收者的消息部分扩散到其他结点。即一个头元素的接收者必须不转发这些头元素到 SOAP

消息路径上的下一个应用程序。同样，接收者可能插入一个相似的头元素。SOAP actor 属性可以用于指示头元素的接收者。SOAP actor 属性的值是一个 URI，用于将 SOAP 接收方结点的名称标识为最终目标位置。下述代码演示了 actor 属性的使用。

```
<soap-env:Envelope xmlns:soap-env="http://schemas.xmlsoap.org/soap/envelope/">
    <soap-env:Header>
        <f:Name xmlns:f="http://www.fruit.com/FruitService/"
        soap-env:actor="http://www.fruit.com/ws"
        soap-env:mustUnderstand="1">
        WebServices
        </f:Name>
    </soap-env:Header>
    <soap-env:Body>
        <m:GetPrice xmlns:m="http://www.fruit.com/prices">
            <m:Item>Apples</m:Item>
        </m:GetPrice>
    </soap-env:Body>
</soap-env:Envelope>
```

另外，SOAP 还有一个特殊的 URI——“http://schemas.xmlsoap.org/soap/actor/next”，通过该 URI 定义的 actor，将使用消息头元素的逐段式（hop-by-hop）通信，在这种通信模型中，SOAP 消息逐一经过多个中间方到达最终目标位置。

注意　SOAP actor 值是用来识别 SOAP 结点的，与路由或消息交换的语义没有联系。

3. SOAP 消息处理

SOAP 消息从发送方到接收方是单向传送，但正如上面所述，SOAP 消息经常以请求/应答的方式实现。SOAP 实现可以通过开发特定网络系统的特性来优化。例如，HTTP 绑定使 SOAP 应答消息以 HTTP 应答的方式传输，并使用同一个连接返回请求。不管 SOAP 被绑定到哪个协议，SOAP 消息采用所谓的“消息路径”发送，这使得在终结点（接收者）之外的中间结点都可以处理消息。一个接收 SOAP 消息的 SOAP 应用程序必须按顺序执行以下的动作来处理消息。

01 识别应用程序需要处理的 SOAP 消息的所有必需内容。

02 检验应用程序是否支持第一步中识别的消息的所有必需内容并处理它。如果不支持，则丢弃消息。

03 在不影响处理结果的情况下，处理器可能忽略第一步中识别出的可选部分。如果这个 SOAP 应用程序不是这个消息的最终目的地，则在转发消息之前删除第一步中识别出来的所有部分。

关于 SOAP 消息处理有以下特点和规定。

- 如果定位到 SOAP 结点的 SOAP 条目有 soap-env:mustUnderstand="1"，但没有被结点理解，则产生一个 SOAP mustUnderstand 错误，如果产生这样的错误，必须停止进一步的处理。
- 如果没有定义 soap-env:mustUnderstand="1"，那么 SOAP 结点可以不处理或忽略该 SOAP 条目。如果一个 SOAP 条目被处理，那么这个 SOAP 结点必须理解该 SOAP 条目而且必须以与该 SOAP 条目说明完全一致的样式进行处理。对于错误，也必须和 SOAP 条目的说明一致。
- 如果 SOAP 结点是 SOAP 中间方，SOAP 消息的样式和处理的结果可以沿着 SOAP 消息路径继续传递。该传递过程以同样的顺序包括来自 SOAP 发送方的所有 SOAP Header 项和 SOAP Body 项。但那些指向 SOAP 中间方的 SOAP Header 项必须被删除。传递过程中，附加的 SOAP Header 项可以插入到 SOAP 消息中。

8.1.4 SOAP 应用模式

为了增强 SOAP 结点之间的通信和消息路径模型，SOAP 选择了一种取决于该通信模型的交互操作模式，如图 8-3 所示。尽管这种模式取决于 SOAP 实现方案，SOAP 消息仍然可以支持下列消息交换模式，以便定义 SOAP 结点（包括中间方）之间的消息路径和传输。

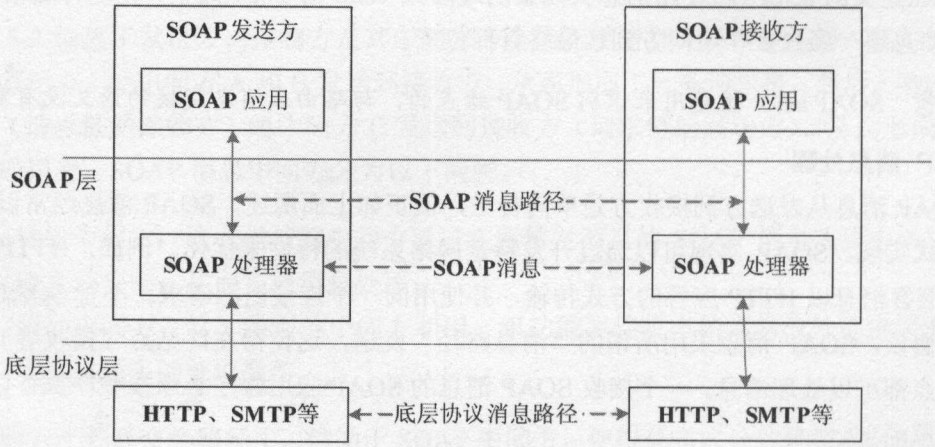


图 8-3 基于底层传输的请求/响应模式

1. 请求/响应模式

请求/响应模式经常用于电子商务交易。该应用模型中，发送方会将一个或多个文档封装在一个请求消息中，然后发送给接收方；接收方处理该消息的内容后响应发送方。请求/响应模式需要通过“请求/响应”消息特性来实现，如图 8-4 所示。

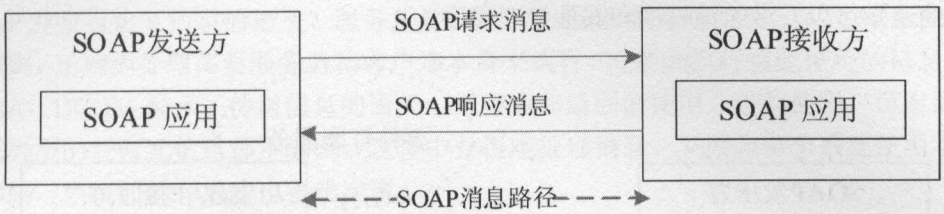


图 8–4　请求/响应模式

2. 多消息异步响应模式

这种模式类似于请求/响应模式，不同的是，应用程序以异步方式向服务器（接收方）发送请求信息，该请求消息在 SOAP 服务器中产生 0 个或多个响应消息并传回客户端(发送方)。这种模式常常在被请求的信息无法一次被完整提供而又要保证较好的系统性能的场合下使用，比如在分布式 Web 搜索这样的场合下应用，如图 8-5 所示。



图 8–5　多消息异步响应模式

3. 单向模式

这种模式也称为“fire-and-forget”，这最初源于军事术语，指武器被发射出去以后就能够自行攻击目标，发射者无须再提供控制。在 SOAP 消息的应用模式中，“fire-and-forget”是指 SOAP 客户应用程序将 SOAP 消息发送给自己的 SOAP 服务器，然后不再去处理与该消息相关的操作。该模式无须返回任何响应，常见于电子邮件消息中，如图 8-6 所示。

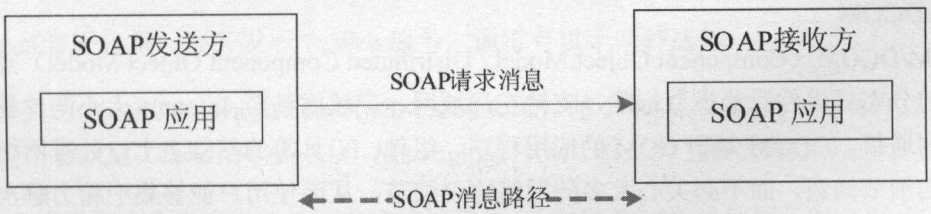


图 8–6　单向模式

4. 事件通知模式

事件通知模式一般发生于订阅情形。应用程序（订阅者，即 SOAP 客户端）向一个事件源（SOAP 服务器）订阅了一些具有明确名称的事件通知，在满足条件的情况下，SOAP 服务器可以将此类事件发送给订阅者及其他应用程序，而无须考虑响应信息。例如一个应用程序可以向打印机驱动程序发出关于打印机状态变化（例如缺纸、缺墨等）的事件请求，而这

些事件的通知可以发送给该程序和其他管理程序。



图 8-7　事件通知模式

8.1.5　SOAP 与 CORBA、COM/DCOM 的区别

CORBA、COM/DCOM 和 SOAP 都是分布式架构，但是它们各有特点。

1. CORBA

CORBA（Common Object Request Broker Architecture）公共对象请求代理体系结构是由 OMG 组织制定的一种标准的面向对象应用程序体系规范。由对象请求代理 ORB、对象服务、公共设施、域接口和应用接口这几个部分组成。其核心部分是对象请求代理 ORB（Object Request Broker）。ORB 提供了一种机制，通过这种机制，对象可以透明地发出请求和接收响应。分布的、可以互操作的对象可以利用 ORB 构造可以互操作的应用。ORB 可看做是在对象之间建立客户/服务关系的一种中间件。基于 ORB，客户可以透明地调用服务对象提供的方法，该服务对象可以与客户运行在同一台机器上，也可以运行在其他机器上通过网络与客户进行交互。ORB 截取客户发送的请求，并负责在该软件总线上找到实现该请求的服务对象，然后完成参数、方法调用，并返回最终结果。

2. COM/DCOM

COM/DCOM（Component Object Model / Distributed Component Object Model）是微软公司提出的分布式组件对象模型标准，支持在局域网、广域网甚至 Internet 上不同计算机的对象之间的通信。DCOM 基于 COM 的应用程序、组件、工具等的基础之上，处理网络协议的低层次的细节问题，而不必关心太多的网络协议细节，从而使用户能够集中精力解决用户所要求的问题。DCOM 位于应用程序的组件之间，将组件以不可见的方式胶合在一起组成具有完整功能的应用程序。

3. SOAP 与 CORBA、COM/DCOM 的比较

首先指出的是 SOAP 不会取代 CORBA 和 COM/DCOM，三者的概念有所区别。COM/DCOM 是一个组件模型标准，CORBA 是分布式应用的服务标准。CORBA 和 DCOM 为分布式应用程序建立服务，服务对象来执行客户端调用的服务。而 SOAP 是基于 XML 和 HTTP 的分布式对象的通信协议，是 COM/DCOM 和 CORBA 对象进行通信的协议。

SOAP 并没有定义信息的语义、服务质量、基于 Internet 的事务处理。而是采用 XML 进行消息编码，正确的处理需要服务器和客户端本身来执行，理解和执行彼此使用的信息格式。而 CORBA，DCOM 表示了传送信息的语义，对参数和返回值使用二进制编码，可对诸如参数名称或类型的任何元信息都不编码，这使中介很难处理消息。又因为每个系统使用不同的二进制编码，系统间的互操作也很难实现。

尽管 CORBA 可以在不同的平台上执行，DCOM 可以在微软的各种平台上运行，但是基于 CORBA 和 DCOM 的解决方案必须依赖于单一的应用程序。假如运行的是 DCOM 服务器程序，所有分布式的客户端不得不运行于微软的操作平台上。CORBA 虽然可以运行于不同的平台，但 CORBA 的互操作性并没有在更高层的服务上进行扩展，如安全性和事务处理，在这种情况下，许多提供的服务没有得到很好的优化。DCOM 和 CORBA 适合于“服务器/服务器”间的通信，但是对于“客户端/服务器”的通信十分脆弱，尤其当客户程序分布在 Internet 上更是如此。

SOAP 不像 DCOM 一样试图定义分布式系统的所有元素，SOAP 没有提供分布式类库，类型安全检查，版本控制等，SOAP 比它处于一个更低的层次，有点儿类似于 IIOP 在 CORBA 的作用，DCOM 却提供了一些额外的协议功能，是 IIOP 或者 SOAP 所不具备的。

简单地理解，SOAP 就是这样的一个开放协议：SOAP=RPC+HTTP+XML，采用 HTTP 作为底层通信协议；RPC 作为一致性的调用途径，XML 作为数据传送的格式，允许服务提供者和服务客户经过防火墙在 Internet 进行通信交互。

DCOM 是有效、灵活的，但也是很复杂的。而 SOAP 的一个主要优点就在于它的简单性，SOAP 使用 HTTP 作为网络通信协议，接收和传送数据参数时采用 XML 作为数据格式，从而代替了 DCOM 中的 NDR 格式，SOAP 和 DCOM 执行过程是类似的，但是用 XML 取代 NDR 作为编码表现形式，提供了更高层次上的抽象，与平台和环境无关。

8.2　JAXM 元素

在 JavaEE 平台里，要开发一个 Web 服务，通常有以下两种选择。

- 使用 JAX-RPC（Java API for XML-based RPC，基于 XML RPC 的 Java API）。
- 使用 JAXM（Java API for XML Messaging，XML 消息的 Java API）。

JAXM 提供了一个用于支持使开发者能够用 Java 实现 SOAP 通信服务的 API 和规范，JAXM 是 JWSDP 的一部分，支持 SOAP1.1 规范。通常我们说的 JAXM API，它包括以下两个包。

- javax.xml.soap：它是发送 SOAP 消息的基本包，主要包含了发送带有附件的 SOAP 消息的 API（SOAP with Attachments API for Java ，SAAJ），为构建 SOAP 包和解析 SOAP 包提供了重要的支持。SAAJ 适合基于文档的同步或者异步通信。
- javax.xml.messaging：定义了 JAXM 的规范，包含了发送和接收消息所需的 API。

注意　JAXM 的理念是通过提供消息传递和 SOAP API，允许开发人员根据 SOAP 编写支持消息传递标准的业务应用程序。随着 JAXM 1.1 版的推出，SOAP API (javax.xml.soap)被分割成了 SAAJ1.1 规范和 JAXM1.1，JAXM1.1 只包含基于消息传递的 API（javax.xml.messaging）。目前，正在使用的 SAAJ 版本是 1.2，SAAJ 1.2 API 主要由 javax.xml.soap 包组成，它为带有多用途互联网邮件扩展协议(MIME)附件的 SOAP 消息提供抽象。该 API 提供了创建到端点的点到点连接的方法、创建并处理 SOAP 消息和附件的方法、以及接收和处理 SOAP 错误的方法。

8.2.1　SOAPElement

javax.xml.soap.SOAPElement 接口是 JAXM DOM API 中的一个关键接口，是 JAXM API 中大部分 SOAP 对象的直接超级接口，SOAPElement 接口的许多方法、属性和 org.w3c.dom.Element 接口是一样的。

SOAP 消息由信封、消息报头（可选）、消息主体和附件等组件组成，同样，在 JAXM API 中提供了 SOAPEnvelope、SOAPHeader、SOAPBody 以及类似的接口来表示这些组件。这些接口都是 SOAPElement 的子接口，而且操作和读取这些组件的大部分工作都是通过 SOAPElement 接口的方法来实现。

注意　由于历史遗留原因，JAXM 的 DOM 实现和 JAXP 使用的 W3C DOM 实现之间有一些相似之处，读者应注意对比参考。

1. 属性

SOAPElement 接口提供了几个添加、检索和删除元素属性的方法，如表 8-1 所示。

表 8-1　SOAPElement 的属性操作方法

方法名	功能说明
SOAPElement addAttribute(Name name, String value)	将带有指定名称和值的属性添加到此 SOAPElement 对象，name 用于指定本地名称或者命名空间限定的名称
java.util.Iterator getAllAttributes()	返回检索元素所有属性的迭代器，通过迭代器返回的每个元素将是 Name 类型的对象
java.lang.String getAttributeValue(Name name)	通过本地或命名空间限定名称来检索属性值
boolean removeAttribute(Name name)	删除带有指定名称的属性

2. 子元素

JAXM DOM 实现也提供了一组添加、检索、删除 SOAPElement 子元素的方法，如表 8-2 所示。

表 8-2　SOAPElement 的常用方法

方法名	功能说明
SOAPElement addChildElement(Name name)	创建使用给定 Name 对象初始化的新 SOAPElement 对象，并将该新元素添加到此 SOAPElement 对象

（续表）

方法名	功能说明
SOAPElement addChildElement(SOAPElement element)	将 SOAPElement 作为此 SOAPElement 实例的子级添加
SOAPElement addChildElement(String localName, String prefix, String uri)	创建使用指定本地名称、前缀和 URI 初始化的新 SOAPElement 对象，并将该新元素添加到此 SOAPElement 对象
java.util.Iterator getChildElements()	返回该元素所有子元素的迭代器列表
void detachNode()	从树中移除此 Node 对象，该方法从 javax.xml.soap.Node 接口继承而来
void recycleNode()	通知 JAXM 实现，该对象不再被使用，对于以后可能创建的结点，实现可以随意重用此对象，该方法只有在调用 detachNode 之后才进行调用
SOAPElement addTextNode(String text)	创建使用给定 String 初始化的新 Text 对象，并将其添加到此 SOAPElement 对象
String getValue()	如果这是一个 Text 结点，则返回此结点的值，否则返回此结点的直接子结点值
void setValue(String value)	如果这是一个 Text 结点则此方法会设置它的值，否则该方法设置此结点的直接（Text）子结点值
void setParentElement(SOAPElement parent)	将此 Node 对象的父结点设置为给定的 SOAPElement 对象
SOAPElement getParentElement()	返回此 Node 对象的父元素

下述代码片段将用于从 SOAP 的主体中删除所有本地名称为“description”的子元素：

```
// soapMsg 是一个 SOAPMessage 对象
SOAPPart part = soapMsg.getSOAPPart();
// 获取 SOAP 信封
SOAPEnvelope env = part.getEnvelope();
// 获取 SOAP 主体
SOAPBody body = env.getBody();
// 创建名为 description 的 Name 对象
Name nm = env.createName("description");
Iterator it = body.getChildElements(nm);
while(it.hasNext()) {
    SOAPElement child = (SOAPElement)it.next();
    child.detachNode();
    child.recycleNode();
}
```

3. 文本结点

文本结点是 JAXM DOM 实现与 JAXP 的另一个不同之处。通过调用 addTextNode 方法

可以把一个文本结点添加到 SOAPElement。通过调用 getValue 方法，可以从文本结点中读取数据。下述代码片段将添加一个带有本地名（fname）和文本结点（apple）的子元素。

```
// soapMsg 是一个 SOAPMessage 对象
SOAPPart part = soapMsg.getSOAPPart();
// 获取 SOAP 信封
SOAPEnvelope env = part.getEnvelope();
// 获取 SOAP 主体
SOAPBody body = env.getBody();
// 创建名为 fname 的元素
SOAPElement child=body.addChildElement("fname");
// 创建值为 apple 的文本结点
child.addTextNode("apple");
// 读取文本值
String str=child.getValue();
```

在执行了上面的代码之后，child 对象将表示为一下 XML 文档：

```
<fname>apple</fname>
```

注意 在调用 getValue 方法读取文本值时，如果直接子元素是一个文本结点，那么该方法返回文本值，否则返回 null。

4. 名称

Name 对象表示 XML 的名称，它可以是本地名称，也可以是命名空间限定的名称。Name 对象需要通过调用 SOAPEnvelope 对象的 createName 方法进行创建，SOAPEnvelope 对象表示 SOAP 消息中的 Envelope。下述代码片段将创建本地和命名空间限定的 Name 对象。

```
// soapMsg 是一个 SOAPMessage 对象
SOAPPart part = soapMsg.getSOAPPart();
// 获取 SOAP 信封
SOAPEnvelope env = part.getEnvelope();
// 创建本地 Name 对象
Name localName = env.createName("model");
// 创建命名空间 Name 对象
Name nsName = env.createName("getPrice","m","http://www.fruit.com/prices");
```

上述代码中，nsName 对象表示带有本地名“getPrice”、前缀“m”以及 URI 为“http://www.fruit.com/prices”的命名空间限定的名称，其在 XML 中表示为：

```
<m:getPrice xmlns:m="http://www.fruit.com/prices">
```

8.2.2 SOAPMessage

javax.xml.soap.SOAPMessage 类是所有 JAXM SOAP 消息的根类，SOAPMessage 对象由一个 SOAP 部分和一个或多个附件部分（可选）组成。SOAPMessage 对象的 SOAP 部分是一个 SOAPPart 对象，包含了用于消息路由和标识的信息，并可以包含特定于应用程序的内容。消息 SOAP 部分中的所有数据都必须是 XML 格式。一般情况下，SOAPMessage 对象包含以下内容。

- 一个 SOAPPart 对象。
- 一个 SOAPEnvelope 对象。
- 一个 SOAPBody 对象。
- 一个 SOAPHeader 对象。

可以通过调用方法 SOAPMessage 的 getSOAPPart 方法检索消息的 SOAP 部分。SOAPEnvelope 对象是从 SOAPPart 对象获取的，SOAPEnvelope 对象用于检索 SOAPBody 和 SOAPHeader 对象，如下述代码所示。

```
SOAPPart sp = message.getSOAPPart();
SOAPEnvelope se = sp.getEnvelope();
SOAPBody sb = se.getBody();
SOAPHeader sh = se.getHeader();
```

除 SOAPPart 对象之外，SOAPMessage 对象还可以包含零个或多个 AttachmentPart 对象，每个 AttachmentPart 对象都包含特定于应用程序的附件数据。SOAPMessage 提供了一些用于创建 AttachmentPart 对象的方法，以及一些将它们添加到 SOAPMessage 对象的方法。

SOAP 消息的附件不需要是 XML 格式的，可以是简单文本或图像文件等任何形式。因此，任何非 XML 格式的消息内容必须在 AttachmentPart 对象中。SOAPMessage 的常用方法如表 8-3 所示。

表 8-3 SOAPMessage 的常用方法

方法名	功能说明
AttachmentPart createAttachmentPart()	创建一个新的空 AttachmentPart 对象
AttachmentPart createAttachmentPart(Object content, String contentType)	创建 AttachmentPart 对象并使用指定内容类型的指定数据填充
AttachmentPart getAttachment(SOAPElement element)	返回与此 SOAPElement 引用的附件关联的 AttachmentPart 对象，如果不存在此类附件，则返回 null
MimeHeaders getMimeHeaders()	返回此 SOAPMessage 对象所有特定于传输的 MIME 头
SOAPBody getSOAPBody()	获取此 SOAPMessage 对象中包含的 SOAPBody
SOAPHeader getSOAPHeader()	获取此 SOAPMessage 对象中包含的 SOAPHeader

（续表）

方法名	功能说明
SOAPPart getSOAPPart()	获取此 SOAPMessage 对象的 SOAP 部分
void removeAllAttachments()	移除所有已经添加到此 SOAPMessage 对象的 AttachmentPart 对象

8.2.3 SOAPPart

javax.xml.soap.SOAPPart 类是 SOAPMessage 对象中特定于 SOAP 部分的容器，所有消息都必须有一个 SOAPPart。消息 SOAP 部分中的所有数据都必须是 XML 格式，因此 SOAPPart 的 MIME 头的 Content-Type 值必须为“text/xml”，不是“text/xml”类型的内容不能在 SOAPPart 对象中，而必须在 AttachmentPart 对象中。

不需要显式地创建 SOAPPart，可以通过调用方法 SOAPMessage.getSOAPPart()方法来获取 SOAPMessage 对象的 SOAPPart 对象。SOAPPart 的常用方法如表 8-4 所示。

表 8-4 SOAPPart 的常用方法

方法名	功能说明
void addMimeHeader(String name, String value)	使用指定名称和值创建 MimeHeader 对象，并将其添加到此 SOAPPart 对象
Iterator getAllMimeHeaders()	返回该 SOAPPart 上的所有 MimeHeader 的迭代器
Source getContent()	以 JAXP Source 对象的形式返回 SOAPEnvelope 的内容
String getContentId()	检索名为“Content-Id”的 MIME 头的值
String getContentLocation()	检索名为“Content-Location”的 MIME 头的值
SOAPEnvelope getEnvelope()	获取与此 SOAPPart 对象关联的 SOAPEnvelope 对象
String[ ] getMimeHeader(String name)	获取此 SOAPPart 对象中所有由给定 String 标识的 MimeHeader 对象的值
void setContent(Source source)	使用给定 Source 对象的数据设置 SOAPEnvelope 对象的内容
void setContentId(String contentId)	将名为“Content-Id”的 MIME 头的值设置为给定 String
void setContentLocation(String contentLocation)	将 MIME 头“Content-Location”的值设置为给定 String
void setMimeHeader(String name, String value)	将与给定头名称匹配的第一个头条目的值更改为给定值，如果不存在相匹配的头，则添加一个带有给定名称和值的新头

8.2.4 SOAPEnvelope

javax.xml.soap.SOAPEnvelope 是 SOAPElement 的子接口，其映射到 SOAPEnvelope 部分。按照 SOAP 规范，SOAPEnvelope 可以包含一个消息报头并且必须包含消息主体，一个 SOAPEnvelope 对象也包含消息报头对象（SOAPHeader）和消息主体对象（SOAPBody），可以通过调用 SOAPEnvelope 的 getHeader()方法和 getBody()方法获取。SOAPEnvelope 的常用

方法如表 8-5 所示。

表 8-5　SOAPEnvelope 的常用方法

方法名	功能说明
SOAPBody addBody()	创建一个 SOAPBody 对象，并将其设置为此 SOAPEnvelope 对象的 SOAPBody 对象
SOAPHeader addHeader()	创建一个 SOAPHeader 对象，并将其设置为此 SOAPEnvelope 对象的 SOAPHeader 对象
Name createName(String localName)	创建使用给定本地名称初始化的新 Name 对象
Name createName(String localName, String prefix, String uri)	创建使用给定本地名称、名称空间前缀和名称空间 URL 初始化的新 Name 对象
SOAPBody getBody()	返回与此 SOAPEnvelope 对象关联的 SOAPBody 对象
SOAPHeader getHeader()	返回此 SOAPEnvelope 对象的 SOAPHeader 对象

8.2.5　SOAPHeader 和 SOAPHeaderElement

javax.xml.soap.SOAPHeader 是 SOAPElement 的一个子接口，其映射到 SOAP 的报头部分。SOAPHeader 的常用方法如表 8-6 所示。

表 8-6　SOAPHeader 的常用方法

方法名	功能说明
SOAPHeaderElement addHeaderElement(Name name)	使用指定名称初始化的新的 SOAPHeaderElement 对象，并将其添加到此 SOAPHeader 对象
Iterator examineHeaderElements(String actor)	返回与特定执行者相匹配的所有报头元素的迭代器
Iterator extractHeaderElements(String actor)	返回在此 SOAPHeader 对象中所有具有指定 actor 的 SOAPHeaderElement 对象的迭代器，并将这些 SOAPHeaderElement 对象从此 SOAPHeader 对象中分离出来

javax.xml.soap.SOAPHeaderElement 表示消息报头内的元素，即 Header 项。SOAPHeaderElement 的常用方法如表 8-7 所示。

表 8-7　SOAPHeaderElement 的常用方法

方法名	功能说明
String getActor()	返回此 SOAPHeaderElement actor 属性的 URL
boolean getMustUnderstand()	返回此 SOAPHeaderElement mustUnderstand 属性的 boolean 值
void setActor(String actor)	将与此 SOAPHeaderElement 对象关联的 actor 设置为指定 actor
void setMustUnderstand(boolean flag)	将此 SOAPHeaderElement 对象的 mustUnderstand 属性设置为指定 boolean 值

8.2.6 SOAPBody 和 SOAPBodyElement

javax.xml.soap.SOAPBody 是 SOAPElement 的一个子接口，表示 SOAP 消息中 SOAP 正文元素的内容。SOAPBody 对象包含了 SOAPBodyElement 对象，后者具有 SOAP 正文的内容，即 Body 项。可以通过调用 SOAPBody 的 addBodyElement()方法创建新的 SOAPBodyElement 对象并将其添加到 SOAPBody 对象。

```
// sbody 是一个 SOAPBody 对象，myName 是一个 Name 对象
SOAPBodyElement sbe = sbody.addBodyElement(myName);
```

SOAPBody 的常用方法如表 8-8 所示。

表 8-8 SOAPBody 的常用方法

方法名	功能说明
SOAPBodyElement addBodyElement(Name name)	使用指定名称创建新的 SOAPBodyElement 对象，并将其添加到此 SOAPBody 对象中
SOAPFault addFault()	创建新的 SOAPFault 对象，并将其添加到此 SOAPBody 对象中
SOAPFault addFault(Name faultCode, String faultString)	使用给定参数创建新的 SOAPFault 对象，并将其添加到此 SOAPBody 对象中
SOAPFault getFault()	返回此 SOAPBody 对象中的 SOAPFault 对象
boolean hasFault()	判断此 SOAPBody 对象中是否存在 SOAPFault 对象

8.2.7 SOAPFault

javax.xml.soap.SOAPFault 以 SOAP 元素作为模型，映射到 SOAP Fault 部分，用于封装 SOAP 消息传递过程中的错误信息。该接口是 SOAPBodyElement 的直接子接口，其常用方法如表 8-9 所示。

表 8-9 SOAPFault 的常用方法

方法名	功能说明
Detail addDetail()	创建新的 Detail 对象，并将其添加到 SOAPFault 对象中，SOAPFault 对象只允许添加一个 Detail 对象
Detail getDetail()	返回此 SOAPFault 对象的 detail 对象
String getFaultActor()	获取此 SOAPFault 对象的错误 actor 的 URI
String getFaultCode()	获取此 SOAPFault 对象的错误代码
String getFaultString()	获取此 SOAPFault 对象的错误描述串
void setFaultActor(String actor)	使用给定值设置 SOAPFault 对象的错误 actor 的 URI
void setFaultCode(String faultCode)	使用给定的错误代码设置此 SOAPFault 对象
void setFaultString(String faultString)	将此 SOAPFault 对象的错误字符串设定为给定字符串

Detail 对象是 DetailEntry 对象的容器。DetailEntry 对象给出特定于应用程序的详细错误信息，以及与包含该对象的 SOAPBody 对象相关的详细错误信息。

可以使用方法 SOAPFault 的 getDetail()方法来检索 Detail 对象（SOAPFault 对象的一部分）。Detail 接口提供两个方法：第一种方法创建一个新 DetailEntry 对象，且自动将其添加到 Detail 对象。第二种方法获取 Detail 对象中包含的 DetailEntry 对象列表。

以下代码片段获取 SOAPFault 对象（sf）的 Detail 对象(detail)，使用 SOAPEnvelope 对象 se 创建 Name 对象并添加到 detail 中，然后获取 detail 中所有 DetailEntry 对象列表。

```
Detail detail = sf.getDetail();
Name name = se.createName("GetLastTradePrice", "WOMBAT",
    "http://www.wombat.org/trader");
detail.addDetailEntry(name);
Iterator it = detail.getDetailEntries();
```

8.3 编写 SOAP 客户端

在基于 SOAP 的 Web Services 体系结构中，客户端可以是一般的 Java GUI 程序（也可以是 JSP、Servlet 等）。客户端通过 SOAP 消息和 Servlet 容器里运行的 JAXM Servlet 进行交互，JAXM Servlet 是服务提供者。

8.3.1 SOAP 连接

JAXM 提供两种类型的连接：

- 消息发送者和接收者之间直接连接。
- 消息发送者和接收者通过消息提供者（Messaging Provider）进行间接连接。

1. 直接连接

直接连接是由消息发送者直接发送给消息接收者的连接，是独立的点对点直接连接。对于此种连接，客户端应用程序无须使用消息提供者，即使不在 Servlet 或者 JavaEE 容器里也能使用。由于其不具有提供者的功能及优点，此类的独立 JAXM 应用程序存在如下限制。

- 独立的应用程序只能点对点发送，单个消息只能直接发送到单个服务。
- 独立的应用程序只能按照请求/响应的模式进行发送。
- 独立的应用程序不能作为 SOAP 服务运行，它们只能作为客户端运行。
- 独立的应用程序只能与运行的服务进行通信，因为它们不访问提供者，所以没有存储和转发机制。

javax.xml.soap.SOAPConnection 表示这种类型的连接。基于点对点连接创建 SOAP 客户端的步骤如下。

01 创建 SOAP 连接。

02 创建 SOAP 消息工厂。
03 创建消息。
04 填充消息。
05 添加附件。
06 发送消息并接收响应。
07 关闭连接。

客户端可以调用 SOAPConnectionFactory 对象的工厂方法获取 SOAPConnection 对象，代码如下所示：

```
SOAPConnectionFactory factory = SOAPConnectionFactory.newInstance();
SOAPConnection connection = factory.createConnection();
```

上述代码顺利执行完毕后，通过调用 SOAPConnection 对象的 call 方法来发送一条同步消息：

```
// 发送 SOAP 消息到目的地
URLEndpoint urlEndpoint = new URLEndpoint(
    "http://localhost:8080/soap/StudentInfoServlet");
// 调用 call 方法发送消息，并返回一个消息
SOAPMessage response = connection.call(message, urlEndpoint);
```

其中，call 方法带两个参数：

- 第一个参数是要发送的 SOAPMessage 对象。
- 第二个参数标识消息发送的目的地，该参数必须是 String、java.net.URL 或 javax.xml.messaging.URLEndpoint 类型的对象。

2. 间接连接

间接连接需要消息提供者、消息发送者和消息使用者通过消息提供者来交互。消息提供者是一种可以提供多种任务的服务型对象，包括提供存储、转发机制来保证异步 SOAP 消息的传送。提供者实际上不是 API 的一部分，它一般作为 Servlet 或某些其他的服务形式存在于 JavaEE 容器中。JAXM API 提供的机制可以使容器托管的应用程序连接到一个提供者，并且把消息发送给它，而提供者将把传入的消息路由到应用程序。

JAXM 提供者只能被运行在容器中的应用程序来使用，这就意味着提供者的实际实现及其功能由特定的厂商来决定。

javax.xml.messaging.ProviderConnection 表示了到消息提供者的连接，该方式需要消息提供者、消息发送者和消息使用者通过消息提供者来交互，通过消息提供者发送消息的基本步骤如下。

01 获取用 JNDI 命名服务注册的提供者连接工厂。
02 使用工厂生成与 JAXM 提供者的连接。

03 使用连接来生成消息工厂。

04 使用消息工厂来生成消息（指定配置文件）。

05 填充消息。

06 使用与 JAXM 提供者的连接来发送消息（异步发送，并且不需等持响应）。

8.3.2 构建消息

由于前面介绍的 JAXM 元素绝大部分是抽象类或接口，不能直接进行实例化，在 JAXM 中提供了一个工厂类——MessageFactory 来生成消息对象以构建可用的 SOAP 消息。

构建 JAXM SOAP 消息的第一步是获得 MessageFactory，可以通过以下两种方式来获取。

- 如果应用程序运行在容器中，并且正使用 JAXM 提供者，则应用程序将使用它与提供者的连接来请求消息工厂。
- 如果应用程序不使用提供者，则可以通过调用 MessageFactory 类的静态方法来返回一个消息工厂的实例。

通过 MessageFactory 创建 SOAP 消息的示例代码如下：

```
// 消息工厂
MessageFactory msgFactory = MessageFactory.newInstance();
// 生成 SOAP 消息实例
SOAPMessage message = msgFactory.createMessage();
```

成功调用 createMessage()方法后，将返回具有基本报头和主体的消息对象，以及封入的信封和 SOAPPart。实际的 SOAP XML 看起来如下：

```
<soap-env:Envelope xmlns:soap-env="http://schemas.xmlsoap.org/soap/envelope/">
<soap-env:Header/>
<soap-env:Body/>
</soap-env:Envelope>
```

创建完 SOAPMessage 对象后，就可以简单地通过消息本身去访问消息对象的各部件。

```
SOAPPart sp = message.getSOAPPart();
SOAPEnvelope se = sp.getEnvelope();
SOAPBody sb = se.getBody();
SOAPHeader sh = se.getHeader();
```

此时，可通过 JAXM API 实现对消息对象的一系列填充操作，如需要添加如下内容的 SOAP 报头内容：

```
<soap-env:Header>
    <s:studentInfo xmlns:s="http://localhost:8080/soap"
        soap-env:mustUnderstand="1">
```

```
        ZKCW9901
    </s: studentInfo>
</soap-env:Header>
```

可以使用如下代码：

```
// 创建名称
Name eName=soapEnvelope.createName("studentInfo ","s",
          "http://localhost:8080/soap");
// 创建并添加新的报头元素
SOAPHeaderElement headerElement= soapHeader.addHeaderElement(eName);
// 设置 mustUnderstand 属性
headerElement.setMustUnderstand(true);
// 添加结点值
headerElement.addTextNode("ZKCW9901");
```

把元素添加到消息主体与把添加到消息报头非常相似，所有操作都必须直接通过 JAXM 的 DOM 实现来完成。

8.3.3 编写 SOAP 客户端

下述代码用于实现任务【描述 8.D.1】，基于消息发送者到接收者的点对点直接连接，创建客户端，向服务器发送 SOAP 消息，并处理服务器的返回消息。

注意 在编写 SOAP 应用时，需将 JAXM 开发工具包 jaxm-api.jar 引入到工程中，或配置到 CLASSPATH。

【描述 8.D.1】 Sender.java

```
public class Sender {
    // 接收方（服务器）地址
    private String serviceurl = "http://localhost:8080/soap/StudentInfoServlet";
    // 消息构建
    public SOAPMessage getMessage() throws SOAPException, Exception {
        // 消息工厂
        MessageFactory msgFactory = MessageFactory.newInstance();
        SOAPMessage message = msgFactory.createMessage();
        // 获得一个 SOAPPart 对象
        SOAPPart soapPart = message.getSOAPPart();
        // 获得信封
        SOAPEnvelope soapEnvelope = soapPart.getEnvelope();
        // 获得消息头
        SOAPHeader soapHeader = soapEnvelope.getHeader();
        // 获得 SOAP 主体
```

```
        SOAPBody soapBody = soapEnvelope.getBody();
        // 创建名称
        Name headerName = soapEnvelope.createName("studentInfo", "stu",
                "http://localhost:8080/soap");
        // 创建并添加新的报头元素
        SOAPHeaderElement headerElement = soapHeader
                .addHeaderElement(headerName);
        // 设置 mustUnderstand 属性
        headerElement.setMustUnderstand(true);
        // 添加结点值
        headerElement.addTextNode("ZKCW9901");
        // 添加消息主体
        Name bodyName = soapEnvelope.createName("getStudentInfo", "stu",
                "http://localhost:8080/soap");
        SOAPBodyElement bodyElement = soapBody.addBodyElement(bodyName);
        // 创建并添加主体元素
        Name eleName = soapEnvelope.createName("StudentName");
        SOAPElement se = bodyElement.addChildElement(eleName);
        se.addTextNode("Zhang Fei");
        // 更新 SOAP 消息
        message.saveChanges();
        return message;
    }
    // 消息发送
    public void send(SOAPMessage message) throws SOAPException, IOException {
        // 创建 SOAP 连接
        SOAPConnectionFactory scf = SOAPConnectionFactory.newInstance();
        SOAPConnection connection = scf.createConnection();
        // 使用 URLEndpoint 构造接收方地址
        URLEndpoint urlEndpoint = new URLEndpoint(serviceurl);
        // 发送 SOAP 消息到目的地，并返回一个消息
        SOAPMessage response = connection.call(message, urlEndpoint);
        if (response != null) {
            // 输出 SOAP 消息到控制台
            System.out.println("Receive SOAP message from localhost:");
            response.writeTo(System.out);
        } else {
            System.err.println("No response received from partner!");
        }
        // 关闭连接
        connection.close();
    }
    public static void main(String[] args) throws SOAPException, Exception {
```

```
        Sender sender = new Sender();
        SOAPMessage message = sender.getMessage();
        sender.send(message);
    }
}
```

上述代码的 getMessage()方法用于构建需要发送的 SOAP 消息，该方法内部按照 XML 文档的结构进行消息的组建。等价的消息内容如下：

```
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">
    <SOAP-ENV:Header>
        <stu:studentInfo xmlns:stu="http://localhost:8080/soap"
         SOAP-ENV:mustUnderstand="1">
            ZKCW9901
        </stu:studentInfo>
    </SOAP-ENV:Header>
    <SOAP-ENV:Body>
        <stu:getStudentInfo xmlns:stu="http://localhost:8080/soap">
            <StudentName>
                Zhang Fei
            </StudentName>
        </stu:getStudentInfo>
    </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

然而，借助于 SOAPPart 的 setContent(Source source)方法，开发人员还可以使用外部文件直接创建消息的 SOAPPart。将上述 SOAP XML 内容存入消息文件（如 message.msg），getMessage()方法即可改写为下述代码。

```
// 消息构建
public SOAPMessage getMessage() throws SOAPException, Exception {
    // 消息工厂
    MessageFactory msgFactory = MessageFactory.newInstance();
    SOAPMessage message = msgFactory.createMessage();
    // 获得一个 SOAPPart 对象
    SOAPPart soapPart = message.getSOAPPart();
    // 从外部文件创建消息
    FileInputStream fis = new FileInputStream("message.msg");
    StreamSource msgSource = new StreamSource(fis);
    // 设置消息内容
    soapPart.setContent(msgSource);
    // 更新 SOAP 消息
    message.saveChanges();
```

```
    fis.close();
    return message;
}
```

本例是遵循请求/响应模式的独立 JAXM 客户端，在服务运行前不能执行。

8.4　编写 SOAP 服务器

SOAP 服务是 SOAP 消息的最终接收者，使用 JAXM，有两种方式可以实现在容器中运行并利用 JAXM 提供者的 SOAP 服务程序。

- 使用 Servlet（在 Servlet 容器中运行）。
- 使用消息驱动 Bean（在 EJB 容器中运行）。

图 8-8 显示了如何使用 Servlet 来实现 SOAP 服务器。可以看出即使在容器内，JAXM 提供者的使用也是可选的。如果通过请求/响应模型与独立的客户端进行通信，那么 SOAP 服务器就不需要使用 JAXM 提供者，Servlet 将通过 Servlet 容器接收来自客户端的请求消息并处理消息，然后把响应消息返回给客户端。另一方面，在使用更复杂的消息路由或使用 JAXM 配置文件的情况，都需要 JAXM 提供者。

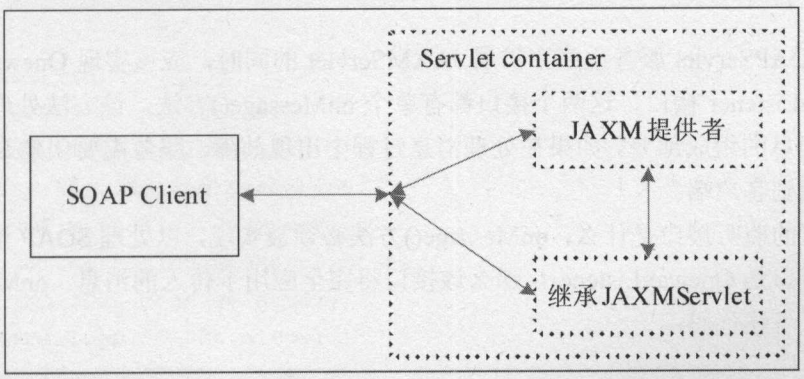


图 8-8　使用 Servlet 实现 SOAP 服务器

8.4.1　JAXMServlet

JAXM API 提供了 javax.xml.messaging.JAXMServlet 用来实现 SOAP 服务器，JAXMServlet 扩展（extend）HttpServlet，是一个抽象类，不能直接实例化该类的对象，必须通过定义该类的子类来实现，同时该子类需实现 javax.xml.messaging.OnewayListener 或 javax.xml.messaging.ReqRespListener 接口：

- javax.xml.messaging.OnewayListener 接口用于定义执行异步 SOAP 通信的 SOAP Servlet 服务器，实现该接口的 Servlet 可以只接收消息而不响应，也可立即响应或隔

一段时间后响应。

- javax.xml.messaging.ReqRespListener 接口用于定义执行同步 SOAP 通信的 Servlet，该 Servlet 将接收消息、处理消息并返回响应消息。

由于 JAXMServlet 扩展自 HttpServlet，一般情况下，扩展 JAXMServlet 的类只需重载 init 方法，以便在 Servlet 初始化时完成相关的准备工作（如，消息工厂初始化等），重载 init 方法的代码类似如下：

```
public class MyJAXMServlet extends JAXMServlet implements ReqRespListener {
    static MessageFactory mf = null;
    public void init(ServletConfig sc) throws ServletException {
        // 初始化父类
        super.init(sc);
        // 创建一个消息工厂
        try {
            mf = MessageFactory.newInstance();
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

另外，SOAPServlet 服务程序在扩展 JAXMServlet 的同时，应该实现 OnewayListener 接口或 ReqRespListener 接口，这两个接口都有单个 onMessage()方法，该方法处理消息、包括访问 SOAP 消息的组成部分。如果在处理消息过程中出现故障，服务需要创建 SOAPFault 对象并把它发回到客户端。

不管选取的监听接口是什么，onMessage()方法必须被实现，以处理 SOAP 消息。如果所选取的监听接口是 OnewayListener，那么该接口将完全应用于传入的消息，onMessage()方法的实现类似于下述代码：

```
public void onMessage(SOAPMessage msg) {
    // 处理传入的请求消息
    // ......
}
```

如果使用 ReqRespListener 实现 Servlet，onMessage()方法需要返回 SOAPMessage 对象，功能实现类似于下述代码：

```
public SOAPMessage onMessage(SOAPMessage msg) {
    // 处理传入的请求消息
    // ......
    SOAPMessage resp = null;
```

```
    // 创建并填充响应消息
    // ......
    // 返回响应消息
    return resp;
}
```

注意　实现 Onewaylistener 的 Servlet 同样可以发出消息，不过这些传出消息将是异步的，它们并不作为传入消息的响应而发回，而是作为未经请求的消息发送到 SOAP 客户端或另一个目的地。

8.4.2　编写 SOAP 服务器

下述代码用于实现任务【描述 8.D.2】，基于消息发送者到接收者的点对点直接连接，创建 SOAP 服务端，接收、处理 SOAP 消息（将 SOAP 消息存入指定文件），并返回 SOAP 响应消息。

注意　在实现 SOAPServlet 时，需将 JAXM 开发工具包 jaxm-api.jar 引入到工程中，或配置到 CLASSPATH，关于 Java Web 应用程序的开发过程此处不做详细介绍，读者可参考相关 Servlet/JSP 的资料。

【描述 8.D.2】 JAXMReceiveServlet.java

```
package soap;
import java.io.File;
import java.io.FileOutputStream;
import javax.servlet.ServletConfig;
import javax.servlet.ServletException;
import javax.xml.messaging.JAXMServlet;
import javax.xml.messaging.ReqRespListener;
import javax.xml.soap.MessageFactory;
import javax.xml.soap.SOAPEnvelope;
import javax.xml.soap.SOAPMessage;
public class JAXMReceiveServlet extends JAXMServlet implements ReqRespListener{
    static MessageFactory mf = null;
    public void init(ServletConfig sc) throws ServletException {
        super.init(sc);
        // 创建一个消息工厂
        try {
            mf = MessageFactory.newInstance();
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
```

```
    // 处理传过来的 SOAP 消息，并返回一个 SOAP 消息
    public SOAPMessage onMessage(SOAPMessage msg) {
        // 返回消息
        SOAPMessage resp = null;
        try {
            // 将 SOAP 请求消息写入指定文件
            FileOutputStream fos =
            new FileOutputStream(new File("c:\\soapmessage.xml"));
            msg.writeTo(fos);
            fos.close();
            // 此处省略返回消息处理的业务逻辑，而用硬编码形式创建一个返回消息
            resp = mf.createMessage();
            SOAPEnvelope se = resp.getSOAPPart().getEnvelope();
            se.getBody().addChildElement(se.createName("ResponseMessage"))
                    .addTextNode("Received Message,Thanks");
            return resp;
        } catch (Exception e) {
            e.printStackTrace();
        }
        return resp;
    }
}
```

该 Servlet 将把接收到的 SOAPMessage 存入到文件中（c:\\soapmessage.xml），为方便演示，采用硬编码方式构造并返回响应消息。

对于该 Servlet，需要通过修改 web.xml 部署文件进行发布，调整后的 web.xml 的内容如下：

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app version="2.4" xmlns="http://java.sun.com/xml/ns/j2ee"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee
    http://java.sun.com/xml/ns/j2ee/web-app_2_4.xsd">
    <servlet>
        <servlet-name>StudentInfoServlet</servlet-name>
        <servlet-class>soap.JAXMReceiveServlet</servlet-class>
        <load-on-startup>1</load-on-startup>
    </servlet>
    <servlet-mapping>
        <servlet-name>StudentInfoServlet</servlet-name>
        <url-pattern>/StudentInfoServlet</url-pattern>
    </servlet-mapping>
</web-app>
```

8.4.3　部署测试

01 首先部署 SOAP 服务器应用到 Web 服务器（如，Tomcat）。

02 启动 Web 服务器，如果启动成功，开始测试。

03 执行 SOAP 客户端应用——sender.java，如果客户端指定的 URI 路径无误，客户端会将消息发送到 SOAP 服务器，并接收 SOAP 服务器应用的响应消息，然后在控制台输出，内容如下（换行符为手工追加）：

```
Receive SOAP message from localhost:
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">
    <SOAP-ENV:Header/>
    <SOAP-ENV:Body>
        <ResponseMessage>
            Received Message,Thanks
        </ResponseMessage>
    </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

04 如果客户端运行通过，此时查看服务器的 C:盘，可看到生成的 soapmessage.xml 文件，该文件的内容即为客户端发送的 SOAP 消息的内容。

小结

通过本章的学习，学生应该能够学会：

- SOAP 是一个基于 XML 的协议，它是分布式系统之间交换信息的轻量级方法。
- SOAP 的两个目标是简单性和可扩展性。
- SOAP 是基于 XML 语言和 XSD 标准的，它由四部分组成：SOAP 信封、SOAP 编码规则、SOAP RPC 表示、SOAP 绑定。
- SOAP 消息可包含如下元素：Envelope（信封）、Header（报头）、Body（主体）、Fault（错误）、attachment（附件），其中 Envelope、Body 部分必须在 SOAPMessage 中出现。
- SOAPEnvelope 是 SOAP 消息结构的主要容器，也是 SOAP 消息的根元素，它必须出现在每个 SOAP 消息中。
- SOAP 不会取代 CORBA，COM/DCOM。
- 在 JavaEE 平台里，要开发一个 Web 服务，通常有两种选择：使用 JAX-RPC 和使用 JAXM。
- javax.xml.soap.SOAPElement 接口是 JAXM API 中大部分 SOAP 对象的直接超级接口。
- 常用的 JAXM 元素有：SOAPElement、SOAPMessage、SOAPPart、SOAPEnvelope、SOAPHeader、SOAPHeaderElement、SOAPBody、SOAPBodyElement、SOAPFault 等。

- JAXM 提供有两种类型的连接：独立的点对点连接和到消息提供者的连接。
- 可通过工厂类 MessageFactory 来生成消息对象以构建可用的 SOAP 消息。
- 编写服务器 Servlet 应用，需扩展 JAXMServlet，并实现 OnewayListener 接口或 ReqRespListener 接口。

练习

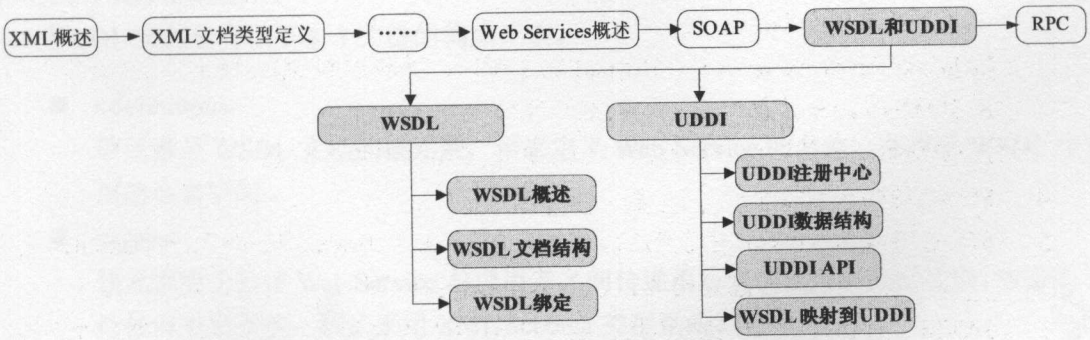
1. 关于 SOAP 消息结构的描述，错误的是_____。
   A. Envelope 元素，必选，可把此 XML 文档标识为一条 SOAP 消息
   B. Header 元素，必选，包含头部信息
   C. Body 元素，必选，包含所有的调用和响应信息
   D. Fault 元素，可选，提供有关处理此消息所发生错误的信息
2. 下列哪些是 SOAP 支持的应用模式_____。（多选）
   A. 请求/响应模式
   B. 多消息异步响应模式
   C. 单向模式
   D. 事件通知模式
3. 下述关于 JAXM 元素描述正确的是_____。（多选）
   A. SOAPElement 是 JAXM API 中大部分 SOAP 对象的直接超级接口
   B. SOAPMessage 类是所有 JAXM SOAP 消息的根类
   C. SOAPPart 类是 SOAPMessage 对象中特定于 SOAP 部分的容器，所有消息都必须有一个 SOAPPart
   D. SOAPEnvelope 是 SOAPElement 的子接口，其映射到 SOAP Envelope 部分
   E. SOAPHeader 映射到 SOAP 的报头部分
   F. SOAPBody 表示 SOAP 消息中 SOAP 正文元素内容的对象
4. SOAP 的两个目标是_____和_____。
5. SOAP 消息可包含如下元素：Envelope、Header、Body、Fault、attachment，其中_____、_____部分必须在 SOAPMessage 中出现。
6. JAXM 提供有两种类型的连接：独立的点对点连接和_____，对于第二种连接需要_____来交互。
7. 使用 Servlet 实现 SOAP 服务器应用，需扩展_____类，并实现_____接口或_____接口。
8. 什么是 SOAP？简要描述 SOAP 的组成。
9. 简要描述基于点对点连接创建 SOAP 客户端的步骤。

第 9 章　WSDL 和 UDDI

本章目标

- 理解 WSDL 的作用
- 熟悉 WSDL 的文档结构
- 了解 WSDL 的 SOAP 绑定
- 了解 UDDI 的作用
- 了解 UDDI 的实现机制
- 了解 UDDI 的数据结构
- 了解 UDDI 的 API
- 理解 WSDL 到 UDDI 数据结构的映射

学习导航



任务描述

【描述 9.D.1】

使用 WSDL 描述天气预报的 Web Service。

【描述 9.D.2】

根据 WSDL 服务接口文档创建 UDDI 的 tModel 结构。

【描述 9.D.3】

根据 WSDL 服务实现文档创建 UDDI 的 businessService 结构。

9.1　WSDL

WSDL（Web Service Description Language，Web Service 描述语言）是一种用来描述 Web Service 的功能特征的语言，主要由 IBM 和微软在 2000 年提出。WSDL 描述了 Web Service 的接口和语义等信息，通过 WSDL，Web Service 的使用者就可以了解这个 Web Service 支持哪些功能调用，以及如何来调用这些功能，进而用户就可以向这个 Web Service 发送 SOAP（或其他协议类型）请求，最终使用这个 Web 服务。

9.1.1　WSDL 概述

Web Service 是一种定义在 Web 上的对象，Web Service 的开发者需要对服务的调用方式进行某种结构化的说明，以便服务的调用者能够正确地使用这些服务。WSDL 就是专门用来描述 Web Service 的一种语言，其规定了一套基于 XML 的语法，能够提供以下关于 Web Service 4 个方面的重要信息。

- 描述服务功能的信息。
- 描述这些功能的传入（请求）和传出（响应）消息的类型信息。
- 描述服务的协议绑定信息。
- 描述用于查找特定服务的地址信息。

WSDL 将 Web Service 定义为端口的集合，一个端口代表一个服务访问点。WSDL 把服务访问点和消息的抽象化描述与具体的服务部署和数据格式的绑定分离，从而使对服务的抽象定义可以方便地重用。

WSDL 文档包含以下 8 个关键的构成元素。

- <definitions>
  该元素是 WSDL 文档的根元素，用来定义 Web Service 的名称，并声明 WSDL 中使用的命名空间。
- <types>
  该元素用于描述 Web Service 与调用者之间传递消息时所使用的数据类型。WSDL 支持任何类型系统，默认采用 XML Schema 类型系统。
- <message>
  该元素是 Web Service 与调用者之间传递的消息的逻辑定义。一个消息可能包含多个部分，每一部分用<part>元素表示，可以使用<types>中定义的数据类型来定义每个<part>的类型。
- <operation>
  该元素是 Web Service 中所支持的操作的抽象定义。通常一个 operation 描述一个服务访问点的请求/响应消息对。

- <portType>
  该元素是某个访问点所支持的所有操作的抽象定义。
- <binding>
  该元素定义了特定<portType>定义的操作和消息的格式、协议之间的绑定。
- <port>
  该元素定义了 Web Service 的绑定地址。
- <service>
  该元素描述了相关的服务访问点的集合。

上述元素中，<types>、<message>、<operation>和<portType>元素是 Web Service 的抽象定义，与具体的 Web Service 部署细节无关，可以被重用；而<binding>、<port>和<service>元素是 Web Service 的具体描述，其中定义了 Web Service 的技术细节。

WSDL 被设计成与语言和平台无关的一种描述语言，其可被用于描述使用任何语言实现的、部署在任何平台上的 Web Service。

9.1.2 WSDL 文档结构

WSDL 文档是一种具有特定含义的 XML 文档，其中使用一些固定的元素来描述 Web Service，这些元素在 9.1.1 节已经介绍过，其具体关系如图 9-1 所示。

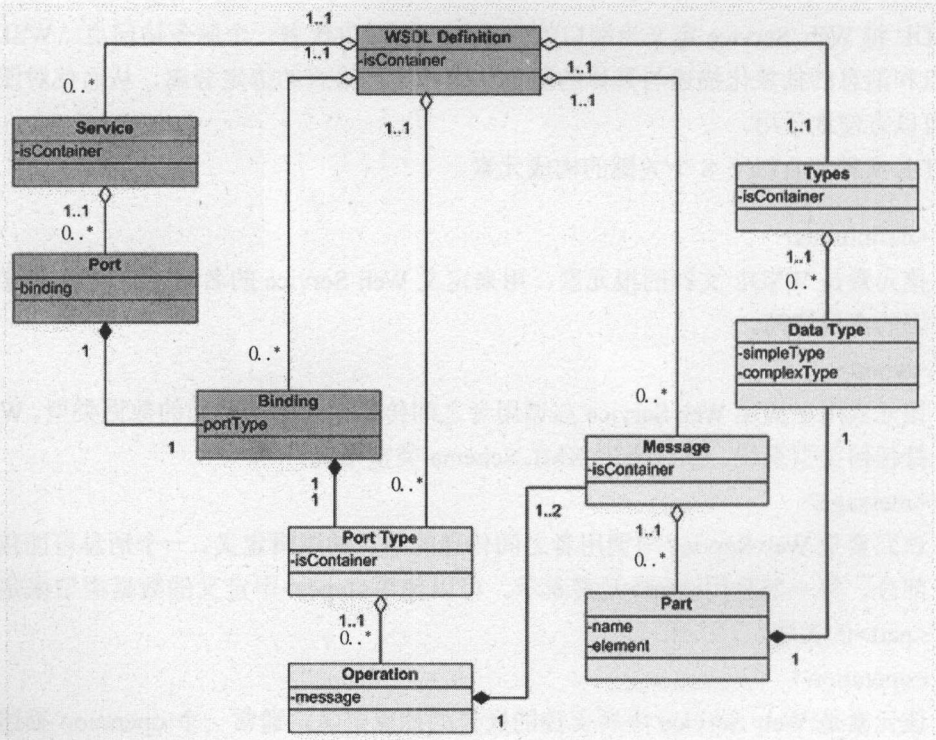


图 9-1 WSDL 文档元素关系

下述代码完成任务【描述 9.D.1】，是天气预报 Web Service 的 WSDL 文档。

【描述 9.D.1】weather.wsdl

```
<?xml version="1.0" encoding="UTF-8"?>
<definitions name="Weather"
    xmlns="http://schemas.xmlsoap.org/wsdl/"
    xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
    xmlns:tns="http://www.example.org/Weather/"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema"
    targetNamespace="http://www.example.org/Weather/">

    <types>
        <xsd:schema targetNamespace="http://www.example.org/Weather/">
            <xsd:element name="WeatherRequest">
                <xsd:complexType>
                    <xsd:sequence>
                        <xsd:element name="city" type="xsd:string" />
                        <xsd:element name="date" type="xsd:date" />
                    </xsd:sequence>
                </xsd:complexType>
            </xsd:element>
            <xsd:element name="WeatherResponse">
                <xsd:complexType>
                    <xsd:sequence>
                        <xsd:element name="temperature" type="xsd:int" />
                        <xsd:element name="humidity" type="xsd:int" />
                    </xsd:sequence>
                </xsd:complexType>
            </xsd:element>
        </xsd:schema>
    </types>

    <message name="getWeatherRequest">
        <part element="tns:WeatherRequest" name="parameters" />
    </message>
    <message name="getWeatherResponse">
        <part element="tns:WeatherResponse" name="parameters" />
    </message>

    <portType name="Weather">
        <operation name="getWeather">
            <input message="tns:getWeatherRequest" />
```

```
            <output message="tns:getWeatherResponse" />
        </operation>
    </portType>

    <binding name="WeatherSOAP" type="tns:Weather">
        <soap:binding style="document"
            transport="http://schemas.xmlsoap.org/soap/http" />
        <operation name="getWeather">
            <soap:operation
                    soapAction="http://www.example.org/Weather/getWeather" />
            <input>
                <soap:body use="literal" />
            </input>
            <output>
                <soap:body use="literal" />
            </output>
        </operation>
    </binding>

    <service name="Weather">
        <port binding="tns:WeatherSOAP" name="WeatherSOAP">
            <soap:address location="http://www.example.org/" />
        </port>
    </service>

</definitions>
```

上述代码中，使用了 WSDL 的各种元素来完整地描述了一个简单的天气预报 Web Service，具体含义如下。

1. <definitions>

<definitions>元素用来定义 WSDL 文档的名称，并且引入需要的 XML 命名空间。weather.wsdl 中的<definitions>元素如下：

```
<definitions name="Weather"
    xmlns="http://schemas.xmlsoap.org/wsdl/"
    xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
    xmlns:tns="http://www.example.org/Weather/"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema"
    targetNamespace="http://www.example.org/Weather/">
```

上述代码中，指定了服务名称为 Weather；然后声明了三个必需的命名空间 WSDL、SOAP 和 XSD，并将 WSDL 命名空间“xmlns="http://schemas.xmlsoap.org/wsdl/"”指定为此 WSDL

文档的默认命名空间，也就是文档中所有没有命名前缀的元素（如<types>、<message>、<portType>、<binding>等）都属于该命名空间；targetNamespace 指定了 WSDL 文档中出现的新元素与属性的命名空间，xmlns:tns 指定了这个命名空间的前缀为 tns。

2. <types>

<types>（类型）元素规定了与消息相关的数据类型的定义。weather.wsdl 中的<types>元素如下：

```
<types>
    <xsd:schema targetNamespace="http://www.example.org/Weather/">
        <xsd:element name="WeatherRequest">
            <xsd:complexType>
                <xsd:sequence>
                    <xsd:element name="city" type="xsd:string" />
                    <xsd:element name="date" type="xsd:date" />
                </xsd:sequence>
            </xsd:complexType>
        </xsd:element>
        <xsd:element name="WeatherResponse">
            <xsd:complexType>
                <xsd:sequence>
                    <xsd:element name="temperature" type="xsd:int" />
                    <xsd:element name="humidity" type="xsd:int" />
                </xsd:sequence>
            </xsd:complexType>
        </xsd:element>
    </xsd:schema>
</types>
```

上述代码中，使用<types>元素定义了 WeatherRequest 和 WeatherResponse 两个复杂类型，其中 WeatherRequest 包含 city 和 date 两个子元素，表示查询某个城市在某一天的天气，WeatherResponse 包含 temperature 和 humidity 两个子元素，表示得到温度和湿度。

3. <message>

<message>（消息）元素定义了传递的消息的数据结构。一个<message>元素由一个或多个<part>（消息片断）元素构成，<part>元素可以使用<types>元素中定义的元素类型。weather.wsdl 中的<message>元素如下：

```
<message name="getWeatherRequest">
    <part element="tns:WeatherRequest" name="parameters" />
</message>
<message name="getWeatherResponse">
    <part element="tns:WeatherResponse" name="parameters" />
```

```
</message>
```

上述代码中，定义了 getWeatherRequest 和 getWeatherResponse 两个<message>元素，分别代表后续的 getWeather 操作的输入和输出消息。<message>元素包含的<part>元素是操作的参数或返回值的名称和类型，即 getWeatherRequest 消息定义了 getWeather 操作的参数名为 parameters，类型为 WeatherRequest，getWeatherResponse 消息定义了 getWeather 操作的返回值名称为 parameters，类型为 WeatherResponse。如果操作使用多个参数或者多个返回值，则<message>元素中可定义多个<part>元素。

4. <portType>

<portType>（端口类型）元素是抽象操作和抽象消息的组合。weather.wsdl 中的<portType>元素如下：

```
<portType name="Weather">
    <operation name="getWeather">
        <input message="tns:getWeatherRequest" />
        <output message="tns:getWeatherResponse" />
    </operation>
</portType>
```

上述代码中，在<portType>元素中组合了 input 和 output 消息，定义了一个名为 getWeather 的操作，即 getWeather 操作的输入消息为 getWeatherRequest，输出消息为 getWeatherResponse。WSDL 支持以下 4 种类型的操作。

- One-way：单向操作。此类型的操作可以接收消息，但不会返回响应，通常由一个<input>元素定义。
- Request-response：请求响应操作。此类型的操作可接收请求消息并返回响应消息，通常由一个<input>元素和一个<output>元素定义，还可以包含一个可选的<fault>元素，用于定义错误消息的抽象数据格式。
- Solicit-response：征求响应操作。此类型的操作可发送请求消息并接收响应消息，通常由一个<output>元素和一个<input>元素定义，也可以包含一个可选的<fault>元素。
- Notification：通知操作。此类型的操作可发送消息，通常由一个<output>元素定义。

5. <binding>

<binding>（绑定）元素用来具体化<portType>元素，其中定义了<portType>元素中的操作和消息的格式与协议等。一个<portType>可以对应多个<binding>。weather.wsdl 中的<binding>元素如下：

```
<binding name="WeatherSOAP" type="tns:Weather">
    <soap:binding style="document"
        transport="http://schemas.xmlsoap.org/soap/http" />
    <operation name="getWeather">
```

```
        <soap:operation
                soapAction="http://www.example.org/Weather/getWeather" />
        <input>
            <soap:body use="literal" />
        </input>
        <output>
            <soap:body use="literal" />
        </output>
    </operation>
</binding>
```

上述代码中，<binding>元素的 type 属性指明了其所绑定的<portType>为 Weather，<soap:binding>元素指明绑定方式为 SOAP，关于 WSDL 绑定的详细介绍见 9.1.3 节。

6. <service>

<service>（服务）元素指定了 Web Service 的位置。一个<service>元素可以包含多个<port>（端口）元素，端口的集合构成了 service。weather.wsdl 中的<service>元素如下：

```
<service name="Weather">
    <port binding="tns:WeatherSOAP" name="WeatherSOAP">
        <soap:address location="http://www.example.org/" />
    </port>
</service>
```

上述代码中，使用<port>元素定义了一个地址，因为在<binding>元素中使用了 SOAP 的绑定方式，所以<port>元素中使用<soap:address>指定了服务的地址为“http://www.example.org/”。

9.1.3　WSDL 绑定

在 WSDL 中，绑定指的是将协议、数据格式与<message>、<operation>、<portType>等抽象实体进行关联的过程。WSDL 支持的 4 种操作类型（单向、请求响应、征求响应、通知）是抽象的定义，而绑定描述了这些抽象概念的具体关联。因此必须为特定的操作类型定义绑定，才能够成功执行该类型的操作。

WSDL 中允许在 WSDL 命名空间定义的元素中使用用户自定义的元素，这些元素称为扩展元素。扩展元素为 WSDL 提供了强大的扩展机制，WSDL 规范中定义了 3 种绑定扩展：

- SOAP 绑定。
- HTTP GET POST 绑定。
- MIME 绑定。

其中 SOAP 绑定是最常用的一种方式。

注意　WSDL 虽然支持 4 种操作类型（单向、请求响应、征求响应、通知），但是 WSDL 规范只对单向和请求响应类型的操作定义了绑定。如果需要描述另外两种类型的操作，那么使用的通信协议必须定义 WSDL 绑定扩展才行。

WSDL1.1 针对 SOAP1.1 定义了绑定扩展。在天气预报 Web Service 的 WSDL 文档中就使用了 SOAP 绑定，weather.wsdl 中关于 SOAP 绑定的代码如下：

```
<binding name="WeatherSOAP" type="tns:Weather">
    <soap:binding style="document"
        transport="http://schemas.xmlsoap.org/soap/http" />
    <operation name="getWeather">
        <soap:operation
                soapAction="http://www.example.org/Weather/getWeather" />
        <input>
            <soap:body use="literal" />
        </input>
        <output>
            <soap:body use="literal" />
        </output>
    </operation>
</binding>
```

上述代码中，使用了 SOAP 命名空间下的<soap:binding>、<soap:operation>和<soap:body>这 3 个元素完成绑定。

<soap:binding>元素规定了采用 SOAP 协议格式的绑定，该元素不提供有关消息格式或编码的信息，但是只要服务使用 SOAP 绑定，就必须声明此元素。

<soap:binding>元素的 style 属性指定了绑定的操作是面向 RPC 还是面向文档（document）的。在面向 RPC 的通信中，消息包含参数和返回值；在面向文档的通信中，消息包含文档，文档的格式没有限制，通常应该是消息的发送者和接收者达成一致的 XML 文档。

<soap:binding>元素的 transport 属性指定了 SOAP 的传输类型，如上述代码中的 transport 属性值“http://schemas.xmlsoap.org/soap/http”代表了 HTTP 传输。不同的值可以代表其他类型的传输，如 SMTP、FTP 等。

<soap:operation>元素定义了 SOAP 操作，其中 soapAction 属性指定了 SOAP 消息报头值。

<soap:body>元素定义了抽象的<part>元素在 SOAP 消息中的具体外观。其中 use 属性指定了消息片断（<part>）使用特定的编码规则还是已定义的具体模式，对应值为“encoded”和“literal”。

注意　第 10 章会详细介绍 RPC 的相关内容。

9.2　UDDI

要调用一个 Web Service 需要几方面的信息。首先需要找到满足需要的 Web Service，还需要了解传送请求的模式，即如何调用这个 Web Service。而 Web 是一个不固定的环境，新的 Web Service 在持续地增加，旧的 Web Service 在不断地删除，已有的 Web Service 的调用方式可能会随时发生变化，所以客观上需要在 Web Service 的发布者和调用者之间建立一种便于查找和发布的机制。UDDI 规范为解决这些问题提供了一种途径，Web Service 的发布者可以将其注册到注册中心，而 Web Service 的调用者可以从注册中心查找到需要的 Web Service 并进行调用。

UDDI（Universal Description，Discovery and Integration，统一描述、发现和集成）是一种基于 Web 的分布式的 Web Service 信息注册中心的实现规范，即它是一种目录服务，企业可以通过它对 Web Service 进行注册和检索。UDDI 技术是 SOAP 和 WSDL 之外的另一项 Web Service 的核心技术。UDDI 可以使提供 Web Service 服务的企业注册服务信息，从而使企业的合作伙伴或潜在的客户能够发现并访问这些 Web Service，也可以使企业发现其他企业提供的服务，以便扩展潜在的业务伙伴关系。通俗地讲，UDDI 相当于 Web Service 的一个公共的注册表，可以理解成电子商务应用与服务的“网络黄页”。因此，UDDI 为企业提供了跨入新市场和服务的机会，使任何规模的企业更为迅速地在全球市场中拓展业务。

9.2.1　UDDI 注册中心

UDDI 规范的实现方案称为 UDDI 注册中心（UDDI Registry），UDDI 注册中心分为公共（public）和私有（private）两种。

公共的 UDDI 注册中心基于 Internet，面向全球企业，任何人都可以在 Internet 上查询或发布商业服务信息。公共注册中心通常包含多个结点，结点之间采用对等网络进行通信，各个结点同步复制了注册信息，因此访问任何一个结点都可以获得与访问其他结点相同的信息。

私有的 UDDI 注册中心由一个组织或一组协作的组织操作，信息只限于成员之间共享。私有的 UDDI 注册中心可以添加更多的安全性控制，以保证 Web Service 注册信息的安全，防止未经授权的用户进行访问。

UDDI 注册中心的数据可以分为以下三类。

- 白页：企业的基本信息，如企业的名称、地址、联系方式、税号等。
- 黄页：根据企业的业务类别来划分的信息类别。
- 绿页：具体描述企业发布的 Web Service 的行为和功能等。

UDDI 注册中心本身也是一个 Web Service，用户可以使用 UDDI 规范中定义的 SOAP API 查询注册信息。

注意 由于各种原因，公共的 UDDI 注册中心并没有广泛地流行。

9.2.2 UDDI 数据结构

UDDI 注册中心存储的信息以 XML 形式表示，称为 UDDI 数据结构，UDDI 规范中定义了这些数据结构的含义以及彼此之间的关系。UDDI 的使用者需要处理的就是这些数据结构，可以使用这些数据结构以及相关的 API 来处理 UDDI 注册中心的信息，对 UDDI 注册中心的搜索结果也是使用这些数据结构来表达，所以 UDDI 数据结构实际上是 UDDI 编程 API 的输入输出参数。

UDDI 数据结构主要包括下列五种元素。

- <businessEntity>：表示商业实体。包含企业的主要信息，如名称、联系方式、根据特定分类法的企业类别、与其他商业实体的关系和特定业务的说明等。
- <publisherAssertion>：表示与其他商业实体的业务关系。比如业务伙伴关系或客户/供应商的关系等。除非双方商业实体都声明了这个业务关系，否则这个业务关系是不对外公开的。
- <businessService>：表示业务服务。这些服务可以是 Web Service 或其他任何类型的服务。<businessService>元素和<bindingTemplate>元素共同构成了“绿页”信息。一个<businessEntity>元素包含一个或多个<businessService>元素，同时一个<businessService>元素可以被多个<businessEntity>元素使用。
- <bindingTemplate>：表示绑定模板。其中包含了指向特定 Web Service 的 URL 和这个 Web Service 的相关技术规范说明。当用户需要调用某个特定的 Web Service 时，必须依据<bindingTemplate>元素中的信息，这样才能保证调用被正确执行。一个<businessService>元素包含了一个或多个<bindingTemplate>元素。
- <tModel>：表示技术模型。其中包含了 Web Service 的分类法或调用规范的引用。一个<bindingTemplate>元素包含了一个或多个<tModel>元素的引用。

上述 5 个 UDDI 核心数据机构的关系如图 9-2 所示。

除了上述 5 个核心元素外，还有以下两个辅助数据结构<identifierBag>和<categoryBag>，用来表示主数据结构的标识信息和类别。

- <identifierBag>：使<businessEntity>和<tModel>包含常见的标识信息，如税号等，这是一个可选的元素，但是如果标识了<identifierBag>，可以极大提高查询 API 的搜索能力。
- <categoryBag>：可根据任何分类机制对<businessEntity>、<businessService>和<tModel>进行分类，这是一个可选的元素，但是如果标识了<categoryBag>，可以极大提高查询 API 的搜索能力。

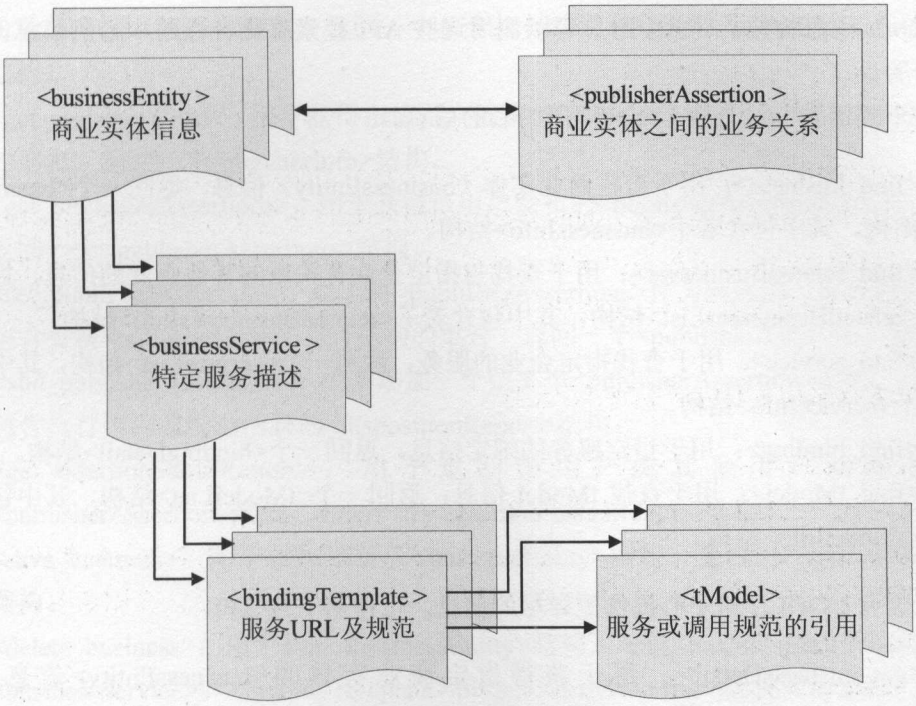


图 9-2　UDDI 核心元素关系

UDDI 的一个重要功能是可以对数据进行分类，这有助于提高搜索效率。UDDI 的<businessEntity>、<businessService>和<tModel>元素可以包含<categoryBag>子元素，其中包含了分类信息。UDDI 内置支持以下三种业务标准分类系统。

- NAICS：北美工业分类系统（North American Industry Classification System）。
- UNSPSC：通用标准产品和服务分类（Universal Standard Products and Services Classification）。
- ISO3166：基于地理位置的标准分类系统。

同时，UDDI 2.0 规范也允许第三方注册新的分类系统。

注意　UDDI 为<businessEntity>、<businessService>、<bindingTemplate>和<tModel>指定了 128 位的通用唯一标识符（UUID），这些标识符可作为访问 UDDI 注册中心中特定数据的 key。

9.2.3　UDDI API

UDDI 规范定义了查询和发布两类 API，可用于与 UDDI 注册中心结点进行通信，这些 API 都是基于 XML 的。

1. 查询 API

查询 API 主要用来从 UDDI 注册中心查找 Web Service 的相关信息，由使用 UDDI Schema

定义的 XML 消息构成。UDDI 用户可以使用这些 API 检索存储在注册中心的信息，查询不需要身份验证。

通过下列函数可以查找 UDDI 注册中心的信息。

- <find_business>：用于查找商业实体（businessEntity）信息，返回一个<businessList>结构，其中包含若干<businessInfo>结构。
- <find_relatedBusinesses>：用于查找与指定企业有关系的其他企业的信息，返回一个<relatedBusinessesList>结构，其中包含若干<relatedBusinessesInfo>结构。
- <find_service>：用于查找指定企业的服务，返回一个<serviceList>结构，其中包含若干<serviceInfo>结构。
- <find_binding>：用于指定服务的绑定信息，返回一个<bindingDetail>结构。
- <find_tModel>：用于查找 tModel 信息，返回一个<tModelList>结构，其中包含若干<tModelInfo>结构。

如果需要 UDDI 注册中心提供更详尽的信息，可使用下列 API。

- <get_businessDetail>：用于获得指定商业实体的<businessEntity>信息，返回<businessDetail>结构。
- <get_businessDetailExt>：用于获得指定商业实体的扩展<businessEntity>信息，返回<businessDetailExt>结构。
- <get_serviceDetail>：用于获得指定服务的<businessService>信息，返回<serviceDetail>结构。
- <get_bindingDetail>：用于获得指定绑定的<bindingTemplate>信息，返回<bindingDetail>结构。
- <get_tModelDetail>：用于获得指定的<tModel>信息，返回<tModelDetail>结构。

以 get_xxx 方式命名的函数可用于查找<businessEntity>、<businessService>、<bindingTemplate>和<tModel>等主要 UDDI 数据结构的详细信息。

2. 发布 API

发布 API 用来对 UDDI 注册中心的数据进行添加、修改和删除。将信息发布到 UDDI 注册中心需要进行身份验证，UDDI 规范没有规定身份验证的机制，这取决于 UDDI 的具体实现，通常使用 HTTPS（Hypertext Transfer Protocol over Secure Socket Layer，安全的超文本传输协议）协议来传递发布调用的请求/响应消息。

通过下列函数可以向 UDDI 注册中心发布信息。

- <get_authToken>：用于向 UDDI 注册中心请求一个身份验证令牌。调用发布 API 中的任何函数，都需要有效的身份验证令牌。返回一个代表令牌的<authToken>结构。
- <discard_authToken>：用于通知 UDDI 注册中心结点丢弃与当前会话关联的活动的身

份验证令牌，相当于注销。返回一个包含执行错误或成功信息的<dispositionReport>结构。

- <get_registeredInfo>：用于获得由该用户管理的全部<businessEntity>和<tModel>文档的列表。返回一个<registerInfo>结构。
- <get_publisherAssertions>：用于获得该用户发布的<publisherAssertions>结构的列表。返回一个<publisherAssertions>结构。
- <set_publisherAssertions>：用于将<publisherAssertions>作为响应的一部分返回，其中包含替换<publisherAssertions>结构的集合。返回一个<publisherAssertions>结构。
- <add_publisherAssertions>：用于添加一个或多个<publisherAssertion>结构。返回一个包含执行错误或成功信息的<dispositionReport>结构。
- <get_assertionStatusReport>：用于返回该用户或其他用户创建的全部<publisherAssertion>列表。返回一个<assertionStatusReport>结构。
- <save_business>：用于添加或更新<businessEntity>结构。返回一个<businessDetail>结构。
- <delete_business>：用于删除<businessEntity>结构。与该<businessEntity>结构关联的<businessService>、<bindingTemplate>和<publisherAssertion>会被同时删除。返回一个包含执行错误或成功信息的<dispositionReport>结构。
- <save_service>：用于添加或更新<businessService>结构。返回一个<serviceDetail>结构。
- <delete_service>：用于删除<businessService>结构。返回一个包含执行错误或成功信息的<dispositionReport>结构。
- <save_binding>：用于添加或更新<bindingTemplate>结构。返回一个<bindingDetail>结构。
- <delete_binding>：用于删除<bindingTemplate>结构。返回一个包含执行错误或成功信息的<dispositionReport>结构。
- <save_tModel>：用于添加或更新<tModel>结构。返回一个<tModelDetail>结构。
- <delete_tModel>：用于删除<tModel>结构。返回一个包含执行错误或成功信息的<dispositionReport>结构。

9.2.4　WSDL 映射到 UDDI

WSDL 是用来描述 Web Service 的一种语言，一个完整的 WSDL 服务描述由一个服务接口和一个服务实现文档组成。WSDL 中的服务接口表示服务的可重用的抽象定义，它在 UDDI 注册中心被作为<tModel>发布；服务实现描述了服务的具体实例，在 WSDL 中表示为<service>元素，每个 WSDL <service>元素都在 UDDI 注册中心发布为<businessService>元素。

当发布一个 WSDL 服务描述时，在服务实现被作为<businessService>发布之前，必须将其对应的服务接口作为<tModel>发布。

WSDL 到 UDDI 的映射关系如图 9-3 所示。

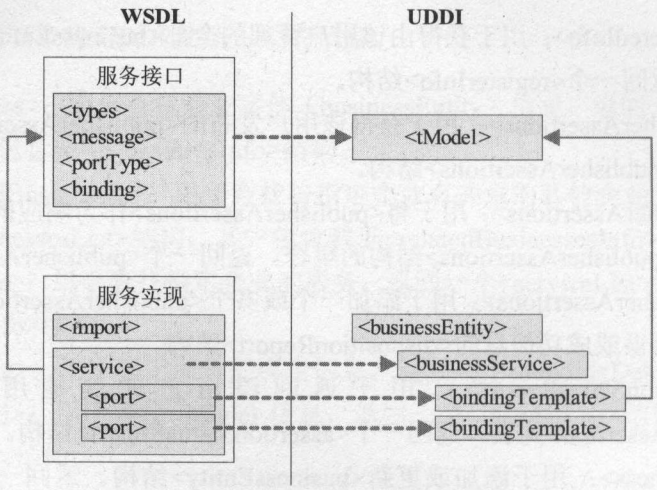


图 9-3　WSDL 到 UDDI 的映射关系

1. 服务接口映射到 tModel

在 UDDI 注册中心，服务接口是作为<tModel>元素由服务接口的提供者发布的。<tModel>元素中的一些子元素是由 WSDL 服务接口描述中的信息构建的，一个有效地对 WSDL 服务接口定义的<tModel>元素引用应该使用 targetNamespace 命名，并且必须包含<overviewURL>和<categoryBag>设置。表 9-1 列出了<tModel>元素的子元素及创建方式。

表 9–1　创建 tModel 的步骤

	UDDI<tModel> 的子元素	WSDL 服务接口	描述	必需
1	<name>	<definitions> 元素的 targetNamespace 属性	<name>元素使用服务接口文档的目标名称空间设置。名称需要一致以确保只使用服务实现文档中的信息就可以定位 tModel	是
2	<description>	<definitions> 元素的 documentation 属性	<description>元素被限制为只能使用 256 个字符。这个元素的值可根据<definitions>元素的前 256 个字符设置。如果<documentation>元素不存在，那么应该使用<definitions>元素中的 name 属性	否
3	<overviewURL>	服务接口文档的 URL 和绑定规范	服务接口文档的位置必须在<overviewURL>元素中设置。如果服务接口文档中有多个绑定，那么必须在 URL 中对绑定进行编码	是
4	<categoryBag>	WSDL 中没有对应信息	<categoryBag>元素必须至少包含一个引用键。这个键必须包含一个对 uddi-org:types tModel 的引用，而且键名必须是“wsdlSpec”。这个条目把 tModel 当做一个 WSDL 服务接口定义	是

下述代码完成任务【描述 9.D.2】，根据 WSDL 服务接口文档创建 UDDI 的 tModel。

【描述 9.D.2】WSDL 接口文档

```
<?xml version="1.0"?>
<definitions name="StockQuoteService-interface"
    targetNamespace="http://www.getquote.com/StockQuoteService-interface"
    xmlns:tns="http://www.getquote.com/StockQuoteService-interface"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema"
    xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
    xmlns="http://schemas.xmlsoap.org/wsdl/">
    <documentation>
        Standard WSDL service interface definition for a stock quote service.
    </documentation>
    <message name="SingleSymbolRequest">
        <part name="symbol" type="xsd:string" />
    </message>
    <message name="SingleSymbolQuoteResponse">
        <part name="quote" type="xsd:string" />
    </message>

    <portType name="SingleSymbolStockQuoteService">
        <operation name="getQuote">
            <input message="tns:SingleSymbolRequest" />
            <output message="tns:SingleSymbolQuoteResponse" />
        </operation>
    </portType>
    <binding name="SingleSymbolBinding"
            type="tns:SingleSymbolStockQuoteService">
        <soap:binding style="rpc"
            transport="http://schemas.xmlsoap.org/soap/http" />
        <operation name="getQuote">
            <soap:operation soapAction="http://www.getquote.com/GetQuote" />
            <input>
                <soap:body use="encoded"
                namespace="urn:single-symbol-stock-quotes"
                encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/>
            </input>
            <output>
                <soap:body use="encoded"
                namespace="urn:single-symbol-stock-quotes"
                encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/>
            </output>
        </operation>
    </binding>
```

```
</definitions>
```

根据表 9-1 中所列的创建方式，创建<tModel>元素，代码如下：

【描述 9.D.2】tModel 结构

```
<tModel tModelKey="">
    <name>http://www.getquote.com/StockQuoteService-interface</name>
    <description xml:lang="en">
        Standard service interface definition for a stock quote service.
    </description>
    <overviewDoc>
        <description xml:lang="en">
            WSDL Service Interface Document
        </description>
        <overviewURL>
    http://www.getquote.com/services/SQS-interface.wsdl#SingleSymbolBinding
        </overviewURL>
    </overviewDoc>
    <categoryBag>
        <keyedReference tModelKey="UUID:C1ACF26D-9672-4404-9D70-39B756E62AB4"
            keyName="uddi-org:types" keyValue="wsdlSpec" />
        <keyedReference tModelKey="UUID:DB77450D-9FA8-45D4-A7BC-04411D14E384"
            keyName="Stock market trading services" keyValue="84121801" />
    </categoryBag>
</tModel>
```

2. 服务实现映射到 businessService

在 UDDI 注册中心，服务的实现是作为带有一个或多个<bindingTemplate>的<businessService>由服务提供者发布的。WSDL 服务实现文档中定义的每个<service>元素都需要创建一个<businessService>元素。

表 9-2 列出了<businessService>元素的子元素及创建方式，这些元素可根据 WSDL 服务实现文档的内容创建。

表 9-2 <businessService>的子元素列表

	UDDI <businessService>的子元素	WSDL 服务实现	描述	必需
1	<name>	<service>元素的 name 属性	<name>元素根据服务实现文档中的<service>元素的 name 属性设置	是
2	<description>	<definitions>元素的 documentation 属性	<description>元素的值根据与<service>元素关联的<documentation>子元素中的前 256 个字符设置。如果<documentation>元素不存在，那么就不创建<description>元素	否

WSDL 文档中的<service>元素代表一个服务的实现，<service>元素的每个<port>元素在 UDDI 中对应一个<bindingTemplate>元素。

表 9-3 列出了<bindingTemplate>元素的子元素及创建方式，这些元素可根据 WSDL 服务实现文档的内容创建。

表 9-3　<bindingTemplate>的子元素列表

	UDDI <bindingTemplate>的子元素	WSDL 服务实现	描述	必需
1	<description>	<definitions> 元素的 targetNamespace 属性	<description>元素的值根据与<port>元素关联的<documentation>子元素中的前 256 个字符设置。如果<documentation>元素不存在，那么就不创建<description>元素	否
2	<accessPoint>	<port> 元素关联的 location 属性	<accessPoint>元素根据与<port>元素关联的扩展元素的 location 属性设置。这个元素将包含 URL，且 URLType 属性是根据此 URL 中的协议设置的。对于不使用 URL 规范的协议绑定，应该使用 URLType 属性指出协议绑定的类型，并且<accessPoint>元素应该包含一个可用于定位使用指定协议的 Web 服务的值	是
3	<tModelInstanceInfo>	WSDL 中没有对应信息	<bindingTemplate>元素包含其引用的每个 <tModel> 元素的一个<tModelInstanceInfo>元素。至少将有一个<tModelInstanceInfo>元素包含对表示服务接口文档的<tModel>元素的直接引用	是
4	<overviewURL>	WSDL 文档地址	<overviewURL>元素可能包含对服务实现文档的一个直接引用，即 WSDL 文件的地址，这只是为了人来读取，这个文档中的其他所有信息都应该能够通过 UDDI 数据实体访问。通过维持对原始 WSDL 文档的直接引用，可以确保被发布的文档就是查找操作返回的文档。如果这个文档包含多个端口，那么这个元素应该包含对端口名的直接引用。由于可能会有多个端口引用同一个绑定，只使用<tModel>元素中的直接引用是不够的。端口名被指定为 overviewURL 上的片段标识符。片段标识符是 URL 的一个扩展，使用“#”字符作为一个分隔符	否

下述代码完成任务【描述 9.D.3】，根据 WSDL 服务实现文档创建 UDDI 的 businessService

结构。

【描述 9.D.3】WSDL 实现文档

```
<?xml version="1.0"?>
<definitions name="StockQuoteService"
    targetNamespace="http://www.getquote.com/StockQuoteService"
    xmlns:interface="http://www.getquote.com/StockQuoteService-interface"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema"
    xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
    xmlns="http://schemas.xmlsoap.org/wsdl/">

    <documentation>
        This service provides an implementation of a standard stock quote service.
        The Web service uses the live stock quote service provided by XMLtoday.com.
        The XMLtoday.com stock quote service uses an HTTP GET interface to request
        a quote, and returns an XML string as a response.
        For additional information on how this service obtains stock quotes,go to
        the XMLtoday.com web site:
        http://www.xmltoday.com/examples/soap/stock.psp.
    </documentation>
    <import namespace="http://www.getquote.com/StockQuoteService-interface"
        location="http://www.getquote.com/wsdl/SQS-interface.wsdl" />
    <service name="StockQuoteService">
        <documentation>Stock Quote Service</documentation>
        <port name="SingleSymbolServicePort"
                binding="interface:SingleSymbolBinding">
            <documentation>Single Symbol Stock Quote Service</documentation>
            <soap:address
                    location="http://www.getquote.com/stockquoteservice"/>
        </port>
    </service>
</definitions>
```

根据表 9-2 及表 9-3 中所列的创建方式，创建<businessService>元素，代码如下：

【描述 9.D.2】businessService 结构

```
<businessService businessKey="..." serviceKey="...">
    <name>StockQuoteService</name>
    <description xml:lang="en">
        Stock Quote Service
    </description>
    <bindingTemplates>
```

```
        <bindingTemplate bindingKey="..." serviceKey="...">
            <description>
                Single Symbol Stock Quote Service
            </description>
            <accessPoint URLType="http">
                http://www.getquote.com/singlestockquote
            </accessPoint>
            <tModelInstanceDetails>
                <tModelInstanceInfo tModelKey="[服务实现对应的 tModel key]">
                    <instanceDetails>
                        <overviewURL>
                            http://www.getquote.com/services/SQS.wsdl
                        </overviewURL>
                    </instanceDetails>
                </tModelInstanceInfo>
            </tModelInstanceDetails>
        </bindingTemplate>
    </bindingTemplates>
    <categoryBag>
        <keyedReference tModelKey="UUID:DB77450D-9FA8-45D4-A7BC-04411D14E384"
            keyName="Stock market trading services" keyValue="84121801" />
    </categoryBag>
</businessService>
```

小结

通过本章的学习，学生应该能够学会：

- WSDL 是专门用来描述 Web Service 的一种语言。
- WSDL 文档包含 8 个关键的构成元素<definitions>、<types>、<message>、<operation>、<portType>、<binding>、<port>和<service>。
- WSDL 绑定指的是将协议、数据格式与<message>、<operation>、<portType>等抽象实体进行关联的过程。
- WSDL 绑定 SOAP 是最常见的绑定方式。
- UDDI（Universal Description，Discovery and Integration，统一描述、发现和集成）是一种基于 Web 的分布式的 Web Service 信息注册中心的实现规范。
- UDDI 注册中心的数据分为白页、黄页和绿页。
- UDDI 数据机构主要包括<businessEntity>、<publisherAssertion>、<businessService>、<bindingTemplate>和<tModel>五种元素。
- UDDI 规范定义了查询和发布两类 API，可用于与 UDDI 注册中心结点进行通信

- WSDL 和 UDDI 的数据结构具有映射关系。

练习

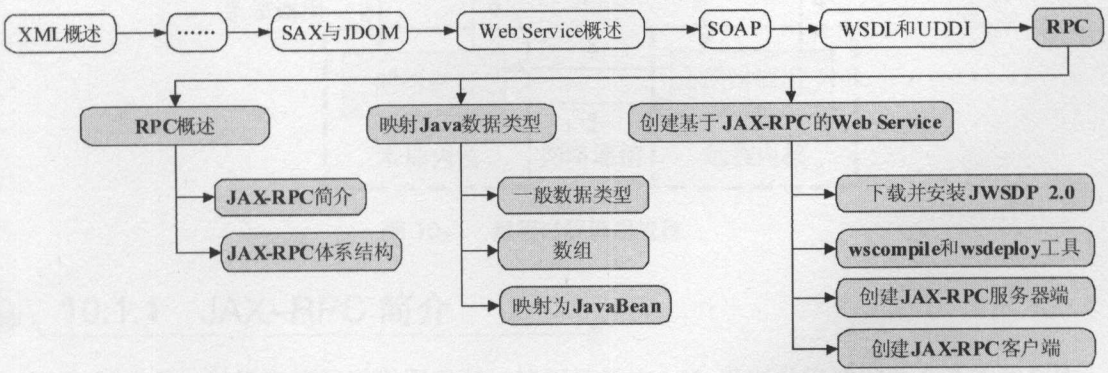
1. 下面不是 WSDL 文档结构的关键要素是：_____。
   A. <definitions>
   B. <types>
   C. <message>
   D. <output>
2. 下面是 WSDL 文档结构的关键要素是：_____。
   A. <portType>
   B. <binding>
   C. <service>
   D. <input>
3. UDDI 注册中心的数据可以分为三类，分别是：____、_____和_____。
4. 简述 UDDI 的功能及特点。

第 10 章　RPC

本章目标

- 了解 JAX-RPC 应用程序体系结构
- 了解基于 JAX-RPC 服务器端的实现方案
- 了解基于 JAX-RPC 客户端的实现方案
- 掌握 JAX-RPC 支持的 Java/XML 数据类型的映射
- 掌握 JAX-RPC 客户端的创建
- 掌握 JAX-RPC 服务器端的创建

学习导航



任务描述

【描述 10.D.1】

创建远程接口 IOrderService 和其实现类 OrderServiceImpl，用于实现订单价格的查询。

【描述 10.D.2】

分别使用动态代理、动态调用方式实现基于 JAX-RPC 的客户端。

10.1　RPC 概述

RPC（Remote Procedure Call Protocol，远程过程调用）协议是一种通过网络从远程计算机程序上请求服务，而不需要了解底层网络技术的协议。RPC 协议建立在某些协议（如 TCP 或 UDP）之上，为通信程序之间携带信息数据，在 OSI 网络通信模型中，RPC 跨越了传输层和应用层。RPC 使得开发包括网络分布式程序在内的应用程序更加容易，其采用客户端/服务器模式，请求程序就是一个客户端，而服务提供程序就是一个服务器。首先，调用进程发送一个有进程参数的调用信息到服务进程，然后等待应答信息；在服务器端，进程保持睡眠状态直到调用信息到达为止，当一个调用信息到达，服务器获得进程参数，计算结果，发送答复信息，然后等待下一个调用信息；最后，客户端调用过程接收答复信息，获得进程结果，然后调用执行继续进行。远程过程调用流程如图 10-1 所示。

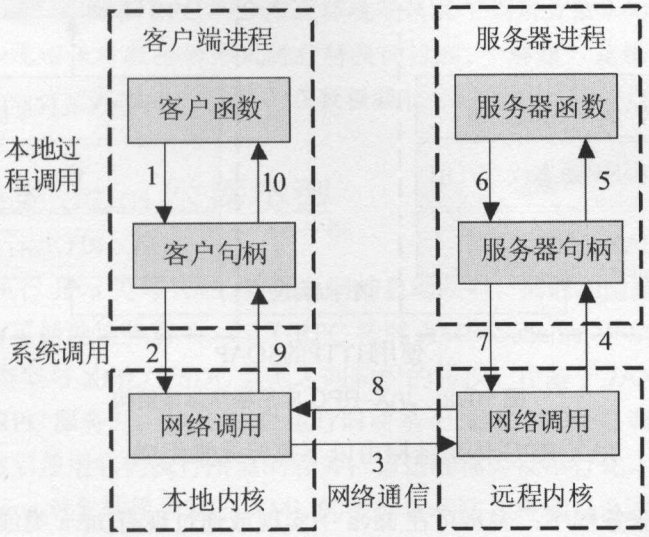


图 10-1　远程过程调用流程

10.1.1　JAX-RPC 简介

JAX-RPC 是一种用来使得远程调用更加简便易行的 API。远程过程调用并不是新鲜事物，该技术已经存在了很长时间，而 JAX-RPC 与过去的 RPC 机制之间的区别在于数据交换格式的不同。与以往的 RPC 实现机制不同，JAX-RPC 的方法调用参数及返回值通常使用 SOAP 消息格式传送，JAX-RPC 使用基于 SOAP 的 RPC 和 WSDL 机制调用异构环境中运行的 Web Service。更重要的是，JAX-RPC 将通过提供 Java 与 XML/WSDL 之间所需的映射，封装了底层 SOAP 包装和消息交换的复杂操作。因此，JAX-RPC 本身是专门用于 XML 消息的，调用 RPC 的客户端和运行 RPC 服务的服务器端以同步方式运行，客户端在得到响应之前处于等待状态，直到 RPC 返回一个响应为止。

目前 JAX-RPC 1.1 完全兼容 SOAP 1.1 协议和 WSDL 规范，并支持 HTTP 作为其主要传

输协议。这样，Java Web Service 开发人员只需要付出少量努力，无须精通 SOAP 或 RPC，即可编写出基于 JAX-RPC 的 Web Service 应用程序。在 Web 服务商业环境中，使用 JAX-RPC，服务请求方和提供方可以执行基于 SOAP 的请求和响应，并以参数或返回值的形式交换 SOAP 消息。

注意 本书使用的是 JAX-RPC 1.1 版本。

10.1.2 JAX–RPC 应用程序体系结构

典型的 JAX-RPC 应用程序体系结构如图 10-2 所示。

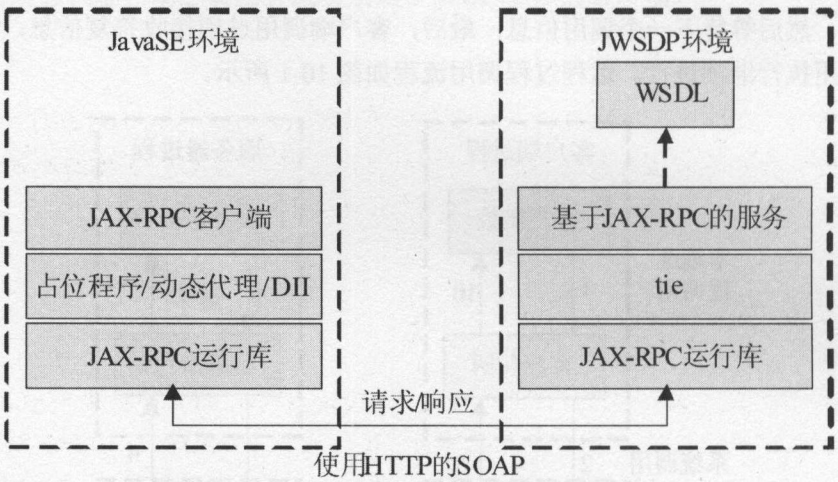


图 10–2 JAX–RPC 应用程序体系结构

如图 10-2 所示，JAX-RPC 体系结构由以下重要元素组成。

- JAX-RPC 服务程序：表示可在 Java 中实现或通过现有 Java 类或 WSDL 文档生成的商业组件。在 JavaEE 环境中，JAX-RPC 服务可实现为 Servlet、无状态会话 bean 或消息驱动 bean。部署期间，JAX-RPC 服务分配到一个或多个服务端点，然后被配置为与传输协议绑定。例如，JAX-RPC 服务可以与 HTTP 绑定，所有消息都使用该服务分配的端点，作为基于 HTTP 的请求和响应交换。JAX-RPC 服务并没有规定只能由 JAX-RPC 客户端访问，因此，在其他平台环境中运行的客户端也可以访问该服务。
- JAX-RPC 客户程序：表示可访问服务的、基于 JAX-RPC 的客户端程序。该客户端程序独立于服务提供方上的目标实现方案。这就意味着被访问的服务可以是使用 Java 平台实现方案的服务，或在非 Java 平台上运行的兼容 SOAP1.1 的服务。为了支持这些客户场景，JAX-RPC 定义了使用不同机制的各种客户调用模型，如基于占位程序的机制、动态代理和动态调用。JAX-RPC 客户端可导入服务提供方发布的 WSDL，并可生成基于 Java 的客户类以访问该服务。
- stub 和 tie：stub 和 tie 都利用核心 JAX-RPC API 来传输 SOAP 编码的 XML。如果使

用 JWSDP（Java Web Services Developer Pack，Java Web 服务开发工具包）来创建一个 Web Service 应用程序，那么 stub 和 tie 都是由 JWSDP 提供的工具来创建的。当创建客户端时将使用生成的 stub 类来调用服务器端上的 Web 服务，这时 stub 充当一个服务代理，从而屏蔽客户端编组和远程过程调用。当创建服务定义和实现时，不需要显示使用 tie 类，JAX-RPC 框架将使用幕后 tie 来解组数据，并调用服务中适当的方法。

- 串行化和反串行化：通信期间，JAX-RPC 使用串行化和反串行化机制进行 Java 与 XML 之间的相互映射，从而实现 Java 基本元素和对象与基于 XML 的表示法之间的转换。还可以为自定义数据类型创建串行器和反串行器。
- JAX-RPC 运行时环境：通过提供所需的运行时库和系统资源，为 JAX-RPC 服务和客户端定义运行时机制和执行环境。

注意　所谓“编组”是指在 RPC 和分布式环境等情况下调用方法和对象时，利用网络传输参数和返回值对数据的形式进行转换的过程。“解组”是编组的逆过程。通常在数据发送方进行数据的编组，在数据接收方进行数据的解组。

10.2　映射 Java 数据类型

JAX-RPC 通过执行 Java 类与 XML 数据之间的自动映射，抽取和隐藏 SOAP 和 XML 数据类型的复杂性。为了处理这些事项，JAX-RPC 按照 XML Scheme 和 SOAP 编码规范，提供 APl 和 Java 数据类型与 XML/WSDL 数据之间映射的转换。在基于 JAX-RPC 的 Web 服务场景中，调用 JAX-RPC 服务时，JAX-RPC 运行时将基于 XML 的 RPC 调用转换为其对应的 Java 对象表示法，然后使用它们执行所需的服务：该过程称为反串行化。执行该操作之后，服务通过将返回的 Java 对象转换为基于 XML 的数据表示法，将调用返回给自己的服务客户端：该过程称为串行化。

10.2.1　一般数据类型

JAX-RPC 规范要求支持大多数的 Java 原始数据类型及原始数据类型对应的包装类型，此外，JAX-RPC 规范还对常用的一些对象类型提供了支持，如：

- java.lang.String
- java.util.Date
- java.util.Calendar
- java.math.BigInteger
- java.math.BigDecimal
- java.lang.Object

JAX-RPC 支持的 Java 基本数据类型与其映射的 XML 数据类型，如表 10-1 所示。

表 10-1　Java 基本类型与 XML 数据类型映射

Java 基本数据类型	XML Scheme 定义
byte	xs:byte
short	xs:short
int	xs:int
long	xs:long
float	xs:float
double	xs:double
boolean	xs:boolean

由表 10-1 中所示的对应关系，对于以下代码：

```
double price;
```

与之相映射的 XML Scheme 的表示法如下所示：

```
<xs:element name="price" type="xs:double"/>
```

JAX-RPC 支持的 Java 对象类型与其映射的 XML 数据类型，如表 10-2 所示。

表 10-2　Java 对象类型与 XML 数据类型映射

Java 对象类型	XML Scheme 定义
java.lang.Integer	xs:int
java.lang.Long	xs:long
java.lang.Float	xs:float
java.lang.Double	xs:double
java.lang.Short	xs:short
java.lang.Boolean	xs:boolean
java.lang.Byte	xs:byte
java.lang.String	xs:string
java.lang.BigDecimal	xs:decimal
java.lang.BigInteger	xs:integer
java.lang.Calendar	xs:dateTime
java.lang.Date	xs:dateTime

由表 10-2 中所示的对应关系，对于以下代码：

```
String name;
```

与之相映射的 XML Scheme 的表示法如下所示：

```
<xs:element name="name" type="xs:string"/>
```

注意 只有 JAX-RPC 支持的数据类型才能在 Web 服务中作为参数传递和返回。

10.2.2 数组

JAX-RPC 将 XML 数组映射为 Java 数组，开发者可以在 WSDL 服务声明中找到几种不同种类的数组声明。

1. 用 restriction 和 wsdl:ArrayType 定义数组

下述代码通过 restriction 使用 WSDL1.1 指定的 wsdl:ArrayType 属性从 soapenc:Array 中派生数组。代码如下所示。

【代码 10-1】String 类型数组

```
<xs:complexType name="ArrayOfString">
    <xs:complexContent>
        <xs:restriction base="soapenc:Array">
            <xs:attribute ref="soapenc:ArrayType" wsdl:ArrayType="xs:string[]">
            </xs:attribute>
        </xs:restriction>
    </xs:complexContent>
</xs:complexType>
```

上述代码利用 XML Schema 的 restriction 方式，通过引用 SOAP 指定的 soapenc:ArrayType 属性，基于 soapenc:Array 类型派生出新的数组，其中，wsdl:ArrayType 类型为 WSDL1.1 中声明的属性，并且该属性决定了数组每个成员的类型。在串行化过程中，JAX-RPC 将这些复杂数组类型映射为 Java 的字符串数组类型 String[ ]。

注意 上述代码中，命名空间前缀“soapenc”和“wsdl”的 URI 分别为“http://schemas.xmlsoap.org/soap/encoding/”和“http://schemas.xmlsoap.org/wsdl/”，想深入了解数组类型请参考：http://www.w3.org/TR/wsdl。

2. 用 restriction 和 soapenc:Array 定义数组

下述代码通过 restriction 并结合 SOAP 规范中声明的 soapenc:Array 类型来派生数组。代码如下所示。

【代码 10-2】String 类型数组

```
<xs:complexType name="ArrayOfString">
    <xs:complexContent>
        <xs:restriction base="soapenc:Array">
```

```
            <xs:sequence>
                <xs:element name="stringArray" type="xs:string"
                maxOccurs="unbounded"/>
            </xs:sequence>
        </xs:restriction>
    </xs:complexContent>
</xs:complexType>
```

上述代码中利用 XML Schema 的 restriction 方式，从 SOAP 的 soapenc:Array 中可以派生出新的数组。其中，unbounded 值表示元素个数不限数量。与第一种定义数组的方式相同，JAX-RPC 将这些复杂数组类型映射为 Java 的字符串数组类型 String[ ]。

10.2.3 映射为 JavaBean

JAX-RPC 支持将 XML 结构和复杂值类型映射为带 getter 和 setter 方法的 JavaBean。该 bean 属性必须是 JAX-RPC 支持的 Java 类型，代码如下所示：

【代码 10- 3】XML 结构

```
<xs:complexType name="Book">
    <xs:sequence>
    <!--最大的作者数目 -->
        <xs:element name="author" type="xs:string"/>
    <!-- 图书价格-->
        <xs:element name="price" type="xs:float"/>
    </xs:sequence>
    <!--图书的 ISDN -->
    <xs:attribute name="isdn" type="xs:string"/>
</xs:complexType>
```

上述代码定义了一个名为 Book 的复杂类型，JAX-RPC 将其映射为类名为 Book 的 JavaBean，代码如下所示：

【代码 10- 4】Book 代码

```
public class Book {
    private String author;//作者
    private float price;//价格
    private String isbn;//ISBN
    public String getAuthor() {
        return author;
    }
    public void setAuthor(String author) {
        this.author = author;
    }
```

```
    public float getPrice() {
        return price;
    }
    public void setPrice(float price) {
        this.price = price;
    }
    public String getIsbn() {
        return isbn;
    }
    public void setIsbn(String isbn) {
        this.isbn = isbn;
    }
}
```

10.3　创建基于 JAX-RPC 的 Web Service

按照 JAX-RPC 1.1 规范，其定义了执行基于 RPC 或直接针对 SOAP 消息的服务和客户实现方案模型。JAX-RPC 的核心类和接口位于包 javax.xml.rpc 中，而 javax.xml.rpc.handler 包则提供了运行时核心类和处理 SOAP 消息的核心类。在创建基于 JAX-RPC 的 Web 服务之前，首先需要下载并安装 JSWDP 2.0 工具包。

10.3.1　下载并安装 JWSDP 2.0

JWSDP 是一个免费的集成工具，它帮助 Java 开发者使用最新的 Web Services 技术和标准建立和测试 XML 程序、Web Services 和 Web 应用程序，JWSDP 当前最新版本是 2.0。JWSDP 可以在 Oracle 官方网站上下载，本书所用 JWSDP 的下载地址为：http://www.oracle.com/technetwork/java/index-jsp-136025.html。

安装 JWSDP 2.0 的具体步骤如下：

01　双击“jwsdp-2_0-windows-i586.exe”程序进行 JWSDP 2.0 的安装，如图 10-3 所示。



图 10-3　欢迎界面

02 单击“下一步”按钮，出现如图 10-4 所示的界面。



图 10-4　接受许可证协议

03 选择“APPROVE”单选项，接受许可证协议，然后单击“下一步”按钮，出现如图 10-5 所示的界面。

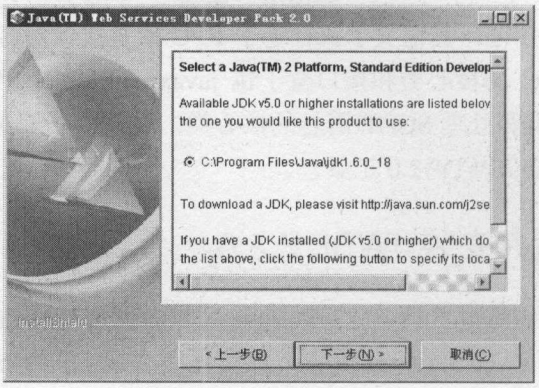


图 10-5　采用的 JDK 版本

04 单击“下一步”按钮，出现如图 10-6 所示的界面。

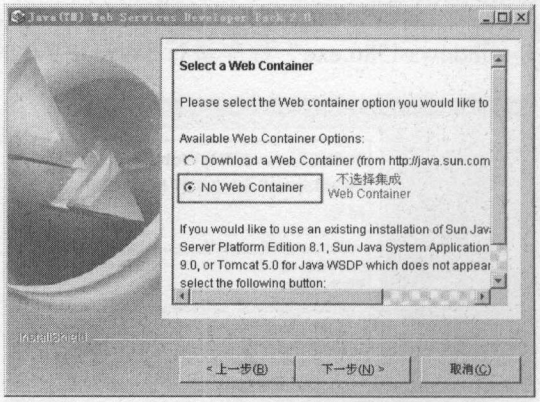


图 10-6　与 Web Container 集成

05 选择“No Web Container”单选项，单击“下一步”按钮，出现如图 10-7 所示的界面。

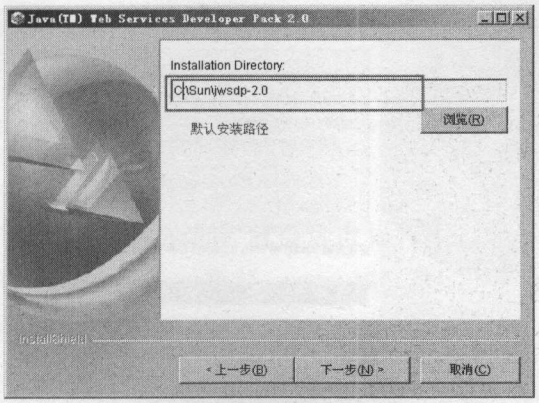


图 10–7　选择安装路径

06 选择安装路径，此处为默认安装路径。单击“下一步”按钮，出现如图 10-8 所示的界面。

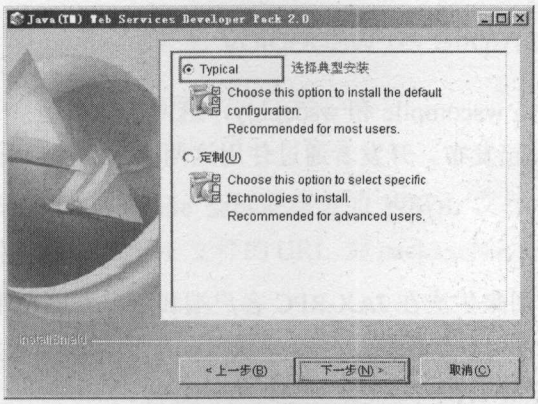


图 10–8　选择安装方式

07 选择安装方式，默认选择“Typical”单选项，单击“下一步”按钮，出现如图 10-9 所示的界面。



图 10–9　安装清单

08 图 10-9 列出了 JWSDP 2.0 提供的组件。单击“下一步”按钮，出现如图 10-10 所示的界面。



图 10-10　安装进度

上图为安装进度图，最后单击“完成”按钮，即可完成 JSWDP 2.0 的安装。

10.3.2　wscompile 和 wsdeploy 工具

JWSDP 的核心工具是 wscompile 和 wsdeploy，这两个工具是由 JAX-RPC 标准实现提供的，它们和 JWSDP 包一起发布。开发者通过使用这两个工具可以开发标准的 Web Services 以及客户端实现。

1. wscompile 工具

wscompile 工具主要用来生成在 JAX-RPC 客户端和服务器端使用的 stubs、ties、serializers 和 WSDL 文件。该工具使用一个配置文件作为输入，配置文件一般指定一个 WSDL 文件、一个模型文件或者一个服务端接口。

wscompile 可以在命令中使用，也可以通过 Ant 脚本调用，命令行中的语法是：

```
wscompile [options] <configuration-file>
```

关于 options 参数的详细信息请参考 jaxrpc-tools 文档，此处不再赘述。

配置文件包含了描述 web service 的信息，基本格式如下：

```
<?xml version="1.0" encoding="UTF-8"?>
<configuration xmlns="http://java.sun.com/xml/ns/jax-rpc/ri/config">
    <service>
        ......省略
    </service>
    或者
    <wsdl>
        ......省略
    </wsdl>
    或者
    <modelfile>
        ......省略
```

```
    </modelfile>
</configuration>
```

其中，<configuration>元素必须包含<service>、<wsdl>或者<modelfile>元素中的一个。

- <service>元素

  如果指定了这个元素，wscompile 就会读取描述服务的终端接口，并且生成一个 WSDL 文件。在<interface>子元素中，name 属性指定了服务终端接口，servantName 属性指定实现接口的类。例如：

```
<service name="服务终端接口名称"
    targetNamespace="目标命名空间"
    typeNamespace="类型的命名空间"
    packageName="包名称">
        <interface name="接口名称" servantName="接口实现类名称"/>
......省略
</service>
```

- <wsdl>元素

  如果指定了该元素，wscompile 会读取服务的 WSDL 文件，产生服务终端接口。其中，location 属性指定了 WSDL 文件的 URL，而 packageName 属性指定了由 wscompile 产生的类所在的包。例如：

```
<wsdl   location="WSDL 文件的 URL"
    packageName="hello">
......省略
</wsdl>
```

- <modelfile>元素

  该元素主要为高级用户而设置。如果 config.xml 文件包含了一个<service>或者<wsdl>元素，wscomplile 就会生成一个 model 文件，它包含了描述服务的内在数据结构。如果已经手动生成了一个 model 文件，那么在下一次运行 wscompile 时可以重用。例如：

```
<modelfile location="mymodel.xml.gz"/>
```

2. wsdeploy 工具

wsdeploy 工具读取一个 WAR 文件和 jaxrpc-ri.xml 文件，然后产生一个可以用来发布的 WAR 文件。在该操作背后，wsdeploy 运行了包含-gen:server 选项的 wscompile。wscompile 命令产生类和一个 WSDL 文件，这些要被包含在 wsdeploy 产生的 WAR 文件中。

和 wscompile 一样，wsdeploy 也可以通过命令行和 Ant 脚本两种方式调用，下面是命令行的语法：

```
wsdeploy -o <output-war-file> <input-war-file> <options>
```

关于 options 参数的详细信息请参考 jaxrpc-tools 文档。

- 输入的 WAR 文件

  开发人员可以通过任何方式（IDE、Ant 等）产生输入的 WAR 文件，这个文件典型结构如下：

```
META-INF/MANIFEST.MF
WEB-INF/classes/hello/HelloIF.class
WEB-INF/classes/hello/HelloImpl.class
WEB-INF/jaxrpc-ri.xml
WEB-INF/web.xml
```

上面这个例子中 HelloIF 是服务终端接口，而 HelloImpl 是实现这个接口的类。web.xml 文件是 web 组件的描述符文件。下面介绍 jaxrpc-ri.xml 文件。

- jaxrpc-ri.xml 文件

  下面的清单是关于一个简单的 HelloWorld 服务的 jaxrpc-ri.xml 文件：

```
<?xml version="1.0" encoding="UTF-8"?>
<webServices
    xmlns="http://java.sun.com/xml/ns/jax-rpc/ri/dd" version="1.0"
    targetNamespaceBase="http://com.test/wsdl"
    typeNamespaceBase="http://com.test/types"
    urlPatternBase="/ws">
        <endpoint name="MyHello"
            displayName="HelloWorld Service"
            description="A simple web service"
            wsdl="/WEB-INF/<wsdlame>"
            interface="hello.HelloIF"
            implementation="hello.HelloImpl"/>
        <endpointMapping  endpointName="MyHello" urlPattern="/hello"/>
</webServices>
```

<webServices>元素必须包含一个或者多个<endpoint>元素。在该例子中，注意<endpoint>元素的 interface 和 implementation 属性指定了服务的接口和实现类。<endpointMapping>元素将服务名和 URL 联系起来。

注意 如果在 CMD 下使用 wsdeploy 命令时，需要配置 PATH 路径。

10.3.3 创建 JAX-RPC 服务器端

JAX-RPC 1.1 规范没有定义用于实现基于 JAX-RPC 的服务的任何 API。在实际开发过程

中通常使用 Java 类来开发 JAX-RPC 服务，开发基于 JAX-RPC 的服务与开发 RMI 应用程序类似，其步骤如下。

01 定义远程接口（服务定义）。
02 实现远程接口（服务实现方案）。
03 配置服务。
04 包装和部署服务。

下面具体讲解以上各步骤。

1. 定义远程接口

远程接口中定义了可由客户端调用的已发布方法的集合，在 JAX-RPC 中，这种方式被称为服务定义，其中，客户占位程序和服务器 tie 就基于该接口生成。定义远程接口时所涉及的编程规则如下所示。

- 远程接口必须扩展 java.xml.Remote 接口，所有方法必须抛出 java.xml.RemoteException 异常。
- 远程接口中不能包含任何静态常量的声明。
- 远程接口中的方法的所有参数和返回值为 JAX-RPC 支持的部分数据类型，否则需要使用自定义的串行器和反串行器进行 Java 与 XML 之间的相互映射。

下述代码用于实现任务【描述 1.D.1】，创建远程接口 IOrderService，并在远程接口中定义用于查询订单价格的方法。

【描述 10.D.1】 IOrderService

```
/**
 * 订单服务接口
 * @author haiersoft
 */
public interface IOrderService extends Remote {

    /**
     * 根据订单编号获取订单价格
     * @param orderNo 订单编号
     * @return 返回订单价格
     * @throws RemoteException 抛出异常
     */
    public String getOrderPrice(String orderNo)throws RemoteException;
}
```

上述代码中，定义了 IOrderService 接口，该接口扩展了 Remote 接口，其中，Remote 接口用于标识其方法可以从非本地虚拟机上调用。任何远程对象都必须直接或间接实现此接

口。只有在远程接口中指定的这些方法才可远程使用。实现类可以实现任意数量的远程接口，并且可以扩展其他远程接口的实现类。

2. 实现远程接口

所谓“远程接口的实现”就是通过自定义类来实现远程接口中定义的所有已发布的方法。

下述代码用于实现任务【描述 1.D.1】，创建远程接口 IOrderService 的实现类 OrderServiceImpl，在该类中编写用于查询订单价格的业务逻辑。

【描述 10.D.1】 OrderServiceImpl

```
/**
 * IOrderService 的实现类
 * @author haiersoft
 *
 */
public class OrderServiceImpl implements IOrderService {
    /**
     * 根据订单编号获取订单价格
     * @param orderNo 订单编号
     * @return 返回订单价格
     * @throws RemoteException 抛出异常
     */
    public String getOrderPrice(String orderNo) throws RemoteException {
        //模拟订单价格
        String orderPrice = "50.25";
        System.out.println("订单号为"+orderNo+"的价格为: "+orderPrice);
        //返回订单价格
        return orderPrice;
    }

}
```

上述代码中，定义了 OrderServiceImpl 类，该类用于实现了 IOrderService 接口，并通过 getOrderPrice()方法模拟了订单价格的查询。

3. 配置服务

要配置服务，需要创建一个 XML 格式的文件，文件中应提供如下信息：

- 服务名称。
- 包含占位程序和 tie 的程序包的名称。
- 生成的 WSDL 及其 XML Scheme 的目标命名空间。
- 远程接口及其实现方案的类名称。

下述代码用于实现任务【描述 10.D.1】，编写用于配置服务的 jaxrpc-ri.xml 文件。

【描述 10.D.1】 jaxrpc–ri.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<webServices xmlns="http://java.sun.com/xml/ns/jax-rpc/ri/dd"
    version="1.0" targetNamespaceBase="http://order.org/wsdl"
    typeNamespaceBase="http://order.org/types" urlPatternBase="/ws">
    <!-- 配置 endpoint，其中 name 为 OrderService -->
    <endpoint name="OrderService" displayName="Order Service"
        wsdl="/WEB-INF/OrderService.wsdl"
        interface="com.haiersoft.ws.IOrderService"
        implementation="com.haiersoft.ws.OrderServiceImpl"/>
    <!-- 配置 endpointMapping，通过服务名称与 URL 联系起来 -->
    <endpointMapping endpointName="OrderService" urlPattern="/order" />
</webServices>
```

上述配置文件应放置于 WEB-INF 目录下。因为在例子中只用了 wsdeploy 工具，所以只需要一个 jaxrpc-ri.xml 文件即可。

4. 打包部署

在项目中添加如图 10-11 所示的包。



图 10–11　添加所需包

其中，jaxrpc-api.jar、jaxrpc-impl.jar 和 jaxrpc-spi.jar 位于 jwsdp-2.0 文件夹下的 jaxrpc\lib 中；saaj-api.jar 和 saaj-impl.jar 位于 jwsdp-2.0 文件夹下的 saaj/lib 中；FastInfoset.jar 则位于 jwsdp-2.0 文件夹下的 fastinfoset\lib 中；mail.jar 和 activation.jar 包位于 jwsdp-2.0 文件夹下的 jwsdp-shared\lib 中。然后，把该工程作为 war 包导出到 C 盘根目录下，导出的名称为：“ch10.war”。然后使用 wsdeploy 工具重新生成 ch10.war 包，命令格式如下：

```
wsdeploy.bat -o d:/ch10.war c:/ch10.war
```

上述命令运行后会将 C 盘下的 ch10.war 重新生成，然后输出到 D 盘根目录下，然后把 D 盘下的 ch10.war 部署到 Tomcat 下。最后访问如下地址：

```
http://localhost:8080/ch10/order?WSDL
```

效果如图 10-12 所示。

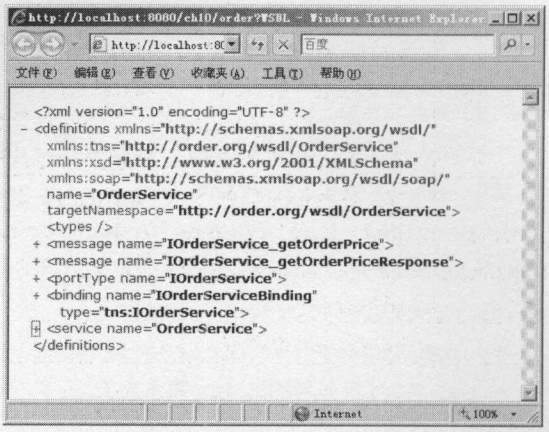


图 10-12　通过 Web 访问 WSDL

上述运行结果是生成的 WSDL 文档，由此可知，基于 JAX-RPC 的 Web 服务部署成功。

10.3.4　创建 JAX-RPC 客户端

根据 JAX-RPC 1.1 规范，基于 JAX-RPC 的客户端程序独立于基于 JAX-RPC 的 Web 服务端。因此，客户端不依赖于其服务提供方或运行的平台是使用 Java 还是非 Java 环境。JAX-RPC 也提供了针对 Java 的客户端 API，并定义了客户端调用模型，从而可以编写 Java 的客户端来访问和调用 Web 服务。

1. JAX-RPC 客户端 API

javax.xml.rpc 程序包提供了一组接口和类，用于创建 JAX-RPC 客户端，在 JAX-RPC 规范中用于创建 JAX-RPC 客户端的接口和类通常有 Stub 接口、Service 接口、Call 接口和 ServiceFactory 类等，分别介绍如下。

- Stub 接口

  java.xml.rpc.Stub 接口是 Stub 类的公共基本接口，所有生成的 Stub 类都要实现该接口。Stub 类的实例表示目标服务端口的客户端代码。

- Service 接口

  java.xml.rpc.Service 接口可以用来获得下述对象。

  - 目标服务端点的动态代理。
  - java.xml.rpc.Call 的实例，用于对目标服务端点动态调用远程操作。
  - Stub 类的实例。

  其主要方法如表 10-3 所示。

表 10-3　Service 接口的方法列表

方法名	功能说明
createCall(QName portName)	创建 Call 实例。其中，参数 portName 是目标服务端点的限定名称
Call[] getCalls(QName portName)	获取预配置的 Call 对象所组成的数组，以便对指定端口调用各种操作。每个对指定端口调用的操作都有一个 Call 对象。每个 Call 对象都是预配置的，不需要使用 Call 接口上的设置方法配置它们。每次调用 getCalls 方法都会返回预配置 Call 对象所组成的新数组。此方法需要 Service 实现类来访问与 WSDL 相关的元数据
Remote getPort(QName portName, Class<T> serviceEndpointInterface)	该方法返回生成的 Stub 实现类的实例或动态代理。服务客户端使用此动态代理对目标服务端点调用各种操作。serviceEndpointInterface 指定创建的动态代理或 Stub 实例所支持的服务端点接口
Qname getServiceName()	获取此服务的名称
URL getWSDLDocumentLocation()	获取此服务的 WSDL 文档的位置

- Call 接口

  java.xml.rpc.Call 接口提供了对服务端点动态调用的支持。其中 Service 接口充当创建 Call 实例的工厂，创建了 Call 实例后，就可以使用各种 getter/setter 方法配置该 Call 实例。Call 接口的主要方法如表 10-4 所示。

表 10-4　Call 接口的方法列表

方法名	功能说明
Object invoke(Object[] inputParams)	使用同步请求-响应交互模式调用特定操作
void invokeOneWay(Object[] inputParams)	使用单向交互模式调用远程方法。等待此远程方法调用的服务器处理完成时，客户端线程通常不会阻塞。当使用的协议为 SOAP/HTTP 时，在收到 HTTP 响应代码或发生错误前，此方法应该阻塞
void removeAllParameters()	从 Call 实例中移除所有的指定参数。注意此方法只移除参数，不移除返回类型。setReturnType(null)用于移除返回类型
void removeProperty(String name)	移除指定属性
void addParameter(String name, QName xmlType, ParameterMode mode)	为特定操作添加参数类型和模式。注意，在调用 invoke 方法之前，客户端代码不能调用任何 addParameter()和 setReturnType()方法。在这种情况下，Call 实现类通过对参数进行反射来确定参数类型

- ServiceFactory 类

  java.xml.rpc.ServiceFactory 类是一个抽象类，提供了用于创建 javax.xml.rpc.Service 对象的工厂。此抽象类遵循抽象静态工厂设计模式，这使客户端能够以可移植方式创建 Service 实例，而无须使用 Service 实现类的构造方法。ServiceFactory 类的方法如表 10-5 所示。

表 10-5　ServiceFactory 类的方法列表

方法名	功能说明
Service createService(URL url,.QName name)	创建 Service 实例，其中，url 的值为 WSDL 的文档位置，name 为服务的 Qname
Service loadService(Class<T> service)	创建用于给定服务接口（如果可用）的已生成服务实现类的实例，其中，参数 service 为服务接口
ServiceFactory newInstance()	获取 ServiceFactory 实例，仅存在一个工厂副本，每次调用此方法时，都会将该副本返回给应用程序。可以通过设置 javax.xml.rpc.ServiceFactory 系统属性重写要使用的实现类

- ParameterMode

  javax.xml.rpc.ParameterMode 是类型安全的参数模式枚举。此类在 Call 的方法中使用，用于指定参数传递模式。ParameterMode 类的方法如表 10-6 所示。

表 10-6　ParameterMode 类的方法列表

属性名	功能说明
ParameterMode.IN	参数传递的 IN 模式
ParameterMode.OUT	参数传递的 OUT 模式
ParameterMode.INOUT	参数传递的 INOUT 模式

2. JAX-RPC 客户端编程模型

JAX-RPC 1.1 规范可以使用下面三种方式的任意一种来开发基于 JAX-RPC 的客户端：

- 占位程序
- 动态调用
- 动态代理

基于占位程序是最简单的一种方式，而在实际应用中主要采用动态调用或动态代理这两种方式来开发 Web 服务客户端，因此下面主要讲解动态代理和动态调用两种方式的应用。

（1）基于动态调用的客户端

JAX-RPC 动态调用接口（Dynamic Invocation Interface，DII）用于访问 Web 服务，该访问方式支持两种类型的调用，即同步的请求-响应方式和单路方式。在请求-响应方式中，客户端利用调用对象的调用方法执行远程方法，然后，客户端等待直到操作完成，即直到响应被返回。在单路方式中，客户端使用调用对象的 invokeOneWay 方法来进行远程调用。使用 DII 开发客户端时主要有以下三个步骤。

01 实例化一个没有 WSDL 的 Service 类。
02 实例化一个 Call 对象代理，并对其进行设置。
03 调用 Call 对象的 invoke 方法。

下述代码用于实现任务【描述 10.D.2】，实现基于动态调用接口的客户端代码编写。

【描述 10.D.2】 DIIClient

```
/**
 * 创建动态调用接口客户端
 *
 * @author haiersoft
 *
 */
public class DIIClient {
    public static void main(String[] args) throws Exception {
        /**
         * 创建一个 Service 实例
         */
        // 服务名称
        String serviceName = "OrderService";
        // 定义服务工厂
        ServiceFactory factory = ServiceFactory.newInstance();
        // 获取服务对象
        Service service = factory.createService(new QName(serviceName));
        /**
         * 创建一个 Call 实例
         */
        //获取 Call 对象
        Call call = service.createCall();
        //命名空间
        String targetNameSpace = "http://order.org/wsdl/OrderService";
        //设置操作名称
        QName operationName = new QName(targetNameSpace,"getOrderPrice");
        call.setOperationName(operationName);
        //设置输入参数
        call.addParameter(
            "String_1",//参数名称 ,注意: wsdl 中发布的参数名称为 String_1
            XMLType.XSD_STRING,//参数在 XML 中的类型
            String.class,//参数在 Java 中类型
            ParameterMode.IN//输入参数
            );
        //设置返回类型
        call.setReturnType(XMLType.XSD_STRING);
        //设置 RPC 风格的操作属性
        call.setProperty(Call.OPERATION_STYLE_PROPERTY,"rpc");
        //soap 编码格式
        String encoding = "http://schemas.xmlsoap.org/soap/encoding/";
        //设置属性
```

```
        call.setProperty(Call.ENCODINGSTYLE_URI_PROPERTY, encoding);
        //终端
        String endpoint = "http://localhost:8080/ch10/order";
        call.setTargetEndpointAddress(endpoint);
        /**
        * 调用 Web 服务端的方法
        */
        String[] params = { "20100503" };
        //返回结果
        String result = (String) call.invoke(params);
        //打印结果
        System.out.println(result);
    }
}
```

上述代码中，首先使用 Service 工厂，并根据已发布的服务名称创建一个 Service 对象，然后根据 Service 对象创建一个 Call 对象，并对 Call 对象设置相关参数和属性，例如命名空间、操作名称等，设置这些属性时，一定要根据已发布的 WSDL 文件设定，否则会出现错误。此外，正如“动态调用接口”其名字所暗示的一样，DII 编程模型是设计用于处理动态服务的，如果服务发生了微小改变，客户端的代码不需要修改太多就能够与之匹配。

上述代码运行结果如下：

```
订单的价格为：50.25
```

注意 在运行客户端的 DIIClient 程序时，请把 JDK 的版本设置为 1.5，由于默认版本为 JDK 1.6，程序会出现下面异常：Exception in thread "main" java.lang.ClassCastException: com.sun.xml.internal.messaging.saaj.soap.ver1_1.Message1_1Impl cannot be cast to com.sun.xml.messaging.saaj.soap.MessageImpl。原因是截止本书出版时 SAAJ 的最高版本为 1.3，而该版本是在 JDK 1.5 环境下运行成功的。

（2）基于动态代理的客户端

动态代理（Dynamic Proxy）客户端可在运行时动态调用目标服务端点，而无须在本地占位程序类。该类型的客户使用 Java 反射 API 提供的动态代理 API。

注意 java.xml.rpc.Service 接口上的 getPort()方法可用于创建动态代理。

与动态调用编程模型类似，使用动态代理开发客户端时主要有以下三个步骤。

01 用 WSDL 实例化一个 Service 对象。
02 实例化一个代理。
03 调用代理的操作。

由上述步骤可知，动态代理与动态调用的步骤 **01**是类似的，不过步骤 **02**和步骤

⁞⁞▸03则大大简化，特别是步骤 ⁞⁞▸02。

下述代码用于实现任务【描述 10.D.3】，实现基于动态代理的客户端的代码编写。

【描述 10.D.3】 DPClient

```
/**
 * 动态代理
 * @author haiersoft
 *
 */
public class DPClient {
    public static void main(String[] args) throws Exception {
        //创建 Service 工厂对象
        ServiceFactory serviceFactory = ServiceFactory.newInstance();
        //WSDL 地址
        String wsdlUrl = "http://localhost:8080/ch10/order?WSDL";
        //服务名称
        String serviceName = "OrderService";
        //命名空间
        String nameSpaceUri = "http://order.org/wsdl/OrderService";
            //根据工厂获取 Service 对象
        Service service = serviceFactory.createService(new URL(wsdlUrl), new
        QName(nameSpaceUri, serviceName));
        //servicePort,注意：与已发布的 wsdl 上的<service>-<port>的 name 对应
        String servicePort = "IOrderServicePort";
        //根据 getPort 方法获取动态代理
        IOrderService orderService = (IOrderService) service.getPort(new QName(
                nameSpaceUri, servicePort),
                IOrderService.class);
        System.out.println("价格为： "+orderService.getOrderPrice("20110502"));

    }
}
```

上述代码中，首先使用 Service 工厂，并根据已发布的服务名称及 WSDL 地址创建一个 Service 对象，然后根据 service 实例的 getPort()方法获取 IOrderService 类型的代理对象，并调用该对象的 getOrderPrice()方法，最终返回运算结果。

运行结果如下：

```
价格为：50.25
```

在调用 Web 服务的众多方法中，最基本的方法是手工生成并发送 SOAP 消息，但这种方法在实际应用中并不常用。上面提到的 DII 方法由于摆脱了 SOAP，因此编程过程相对简单，

不过更好的方法是使用动态代理的方式，该方式利用 WSDL 生成服务端点接口，并利用该接口的代理对象直接调用 Web 服务中已发布的方法，从而大大简化了编程。

小结

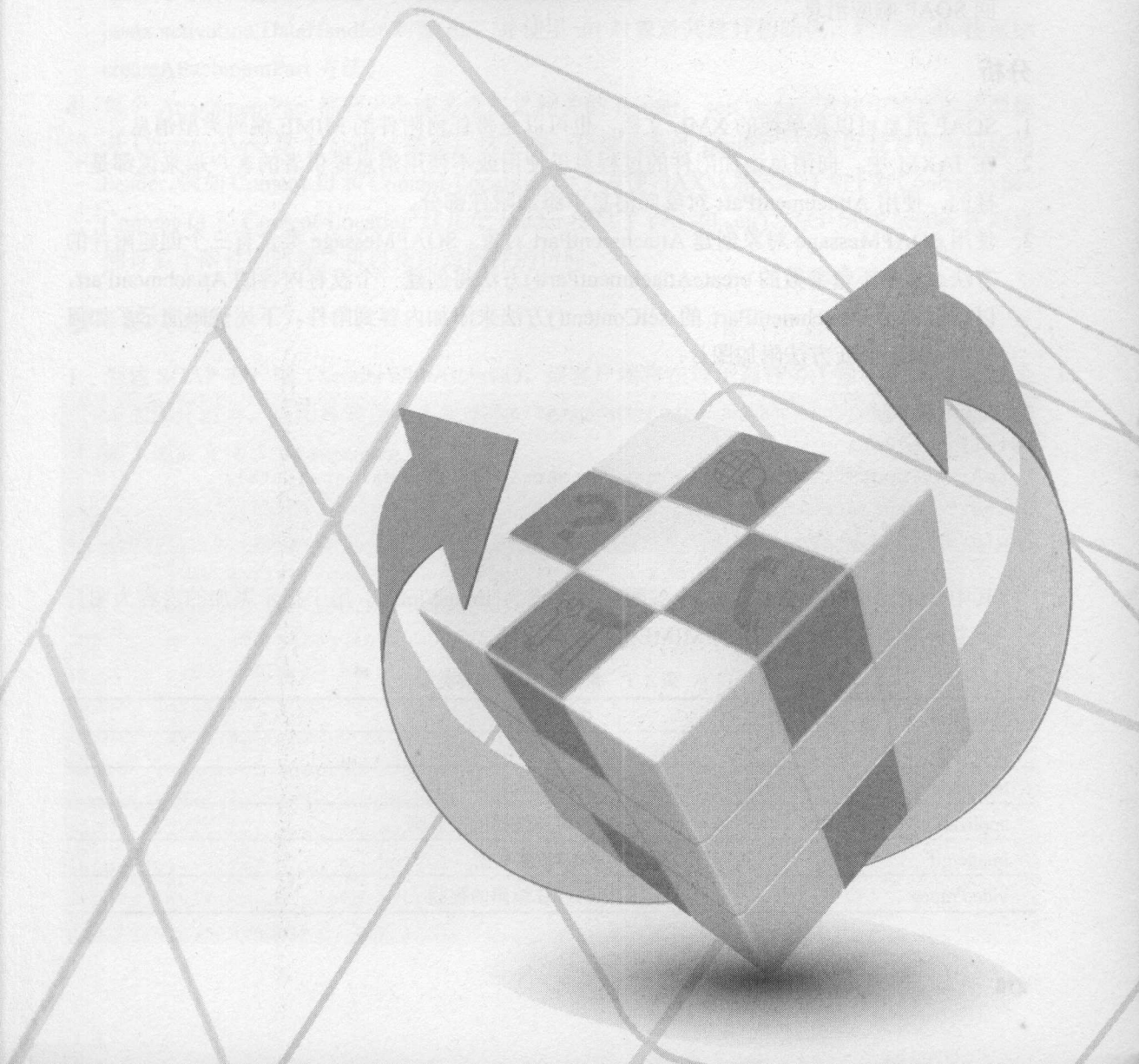
通过本章的学习，学生应该能够学会：

- JAX-RPC 与过去的 RPC 机制之间的区别在于数据交换格式，即 JAX-RPC 的方法调用参数以及返回值通常以 SOAP 消息格式传输。
- JAX-RPC 通过执行 Java 类与 XMI 数据之间的自动映射，抽取和隐藏 SOAP 和 XML 数据类型的复杂性。
- JAX-RPC 规范要求支持大多数的 Java 原始数据类型及原始数据类型对应的包装类型，此外，对 String、Date、Calendar 等对象类型提供了支持。
- JAX-RPC 可以将 XML 数组映射为 Java 数组。
- JAX-RPC 支持将 XML 结构和复杂值类型映射为带 getter 和 setter 方法的 JavaBean。
- wscompile 主要用来生成在 JAX-RPC 客户端和服务中使用的 stubs、ties、serializers 和 WSDL 文件。
- wsdeploy 工具读取一个 WAR 文件和 jaxrpc-ri.xml 文件，然后产生另外一个可以用来发布的 WAR 文件。
- 实现基于 JAX-RPC 的服务有两种方式：使用 Java 类开发 JAX-RPC 服务和使用 WSDL 文档开发 JAX-RPC 服务。
- JAX-RPC 客户端编程模型主要有三种：占位程序、动态调用和动态代理。

练习

1. JAX-RPC 与以往的 RPC 方式的主要区别是_____。
   A. JAX-RPC 是同步的
   B. JAX-RPC 是基于 Java 的
   C. JAX-RPC 传输的数据使用 SOAP 封装
   D. JAX-RPC 基于 HTTP
2. 下述 Java 数据类型可以映射到 XML 数据类型的是_____。（多选）
   A. int　　B. Double　　C. String　　D. Calendar
3. 简述创建 JAX-RPC 服务器端的步骤。
4. 简述创建 JAX-RPC 客户端的步骤。

实践篇



实践 1　SOAP



实践指导

实践 1.G.1

基于消息发送者到接收者的直连，创建 SOAP 应用，具体要求如下。

1. 创建客户端，向服务器发送带附件的 SOAP 消息，并处理服务器的返回消息。
2. 创建 SOAP 服务端，接收、处理 SOAP 消息（将 SOAP 消息及附件存入指定文件），并返回 SOAP 响应消息。

分析

1. SOAP 消息可以是单纯的 XML 文档，也可以是带任何附件的 MIME 编码类型消息。
2. 在 JAXM 中，向消息添加附件的过程对于使用或不使用消息提供者的客户端来说都是一样的，使用 AttachmentPart 对象向消息中添加附件部分。
3. 使用 SOAPMessage 对象创建 AttachmentPart 对象。SOAPMessage 类含有三个创建附件的方法。其中不带参数的 createAttachmentPart()方法将创建一个没有内容的 AttachmentPart，以后再调用 AttachmentPart 的 setContent()方法来添加内容到附件。下述代码演示了如何使用 setContent 方法附加图片：

```
AttachmentPart ap = msg.createAttachmentPart();
byte[] jpegData = ...;
ByteArrayInputStream stream = new ByteArrayInputStream(jpegData);
ap.setContent(stream,"image/jpeg");
msg.addAttachmentPart(ap);
```

其中，setContent 方法的第二个参数必须被设为"image/jpeg"，用于指示添加的内容为图片类型。表 1.1 列出了常用的 MIME 内容类型。

表 1.1　常用 MIME 内容类型

内容类型	内容描述
text/xml	XML 文本
text/plain	纯文本
application/octet-stream	二进制数据的字节流
image/gif	GIF 图像数据
video/mpeg	MPEG 编码的视频

AttachmentPart 的另外两个方法允许创建带内容的 AttachmentPart 对象。其中一个与 AttachmentPart.setContent()方法很像。它需要一个包含内容的 Java 对象和一个指定内容类型的 String 作为参数。对象可以是 String、stream、javax.xml.transform.Source 对象或 javax.activation.DataHandler 对象。

另一个创建带内容的 AttachmentPart 对象的方法需要一个 DataHandler 对象作为输入参数，它是 JavaBeans 激活框架(JAF)的一部分，以下代码演示了如何在内容中使用 DataHandler。

```
URL url = new URL("http://someurl/logo.jpg");
DataHandler dh = new DataHandler(url);
AttachmentPart ap = msg.createAttachmentPart(dh);
ap.setContentId("logo_image")
msg.addAttachmentPart(ap);
```

上述代码中，首先为想要作为附件内容的文件创建一个 java.net.URL 对象 url，然后创建 javax.activation.DataHandler 对象 dh，并使用 url 对象对其进行初始化，然后把 dh 传递给 createAttachmentPart 方法。

4. 每个 AttachmentPart 都有一个或多个与之相关的 header。setContent 方法中用到的类型就是 Content-Type 这个 header 类型。这是唯一必须具备的 header。还可以设置其他可选的 header，例如 Content-Id 和 Content-Location。为了方便，JAXM 和 SAAJ API 为 Content-type、Content-Id 和 Content-Location 这几个 header 提供了 get()和 set()方法。这些 header 在消息拥有多个附件的时候，可以帮助对附件的访问。

参考解决方案

1. 创建 SOAP 客户端（SenderWithAtt.java），该客户端将在理论篇任务【描述 8.D.1】基础上添加图片附件，该图片来源于本地磁盘（baidu.gif 和 yahoo.jpg），同时要发送的消息也来源于磁盘文件（message.msg）。

```
import java.io.FileInputStream;
import java.io.IOException;
import javax.activation.DataHandler;
import javax.activation.FileDataSource;
import javax.xml.messaging.URLEndpoint;
import javax.xml.soap.AttachmentPart;
import javax.xml.soap.MessageFactory;
import javax.xml.soap.SOAPConnection;
import javax.xml.soap.SOAPConnectionFactory;
import javax.xml.soap.SOAPException;
import javax.xml.soap.SOAPMessage;
import javax.xml.soap.SOAPPart;
import javax.xml.transform.stream.StreamSource;
public class SenderWithAtt {
```

```
    private String yahooLogo = "yahoo.jpg";
    private String baiduLogo = "baidu.gif";
    private String serviceurl =
    "http://localhost:8080/soap/StudentInfoServletWithAtt";
    public SOAPMessage getMessage() throws SOAPException, Exception {
        // 消息工厂
        MessageFactory msgFactory = MessageFactory.newInstance();
        SOAPMessage message = msgFactory.createMessage();
        // 获得一个 SOAPPart 对象
        SOAPPart soapPart = message.getSOAPPart();
        // 从外部文件创建消息
        FileInputStream fis = new FileInputStream("message.msg");
        StreamSource msgSource = new StreamSource(fis);
        // 设置消息内容
        soapPart.setContent(msgSource);
        // 使用 DataHandler 封装附件
        DataHandler yahooHandler =
        new DataHandler(new FileDataSource(yahooLogo));
        DataHandler baiduHandler =
        new DataHandler(new FileDataSource(baiduLogo));
        // 创建附件对象
        AttachmentPart yahooAp = message.createAttachmentPart(yahooHandler);
        AttachmentPart baiduAp = message.createAttachmentPart(baiduHandler);
        yahooAp.setContentId("yahoo_logo");
        baiduAp.setContentId("baidu_logo");
        // 向消息中添加附件对象
        message.addAttachmentPart(yahooAp);
        message.addAttachmentPart(baiduAp);
        message.saveChanges();
        return message;
    }
    public void send(SOAPMessage message) throws SOAPException, IOException {
        // 创建 SOAP 连接
        SOAPConnectionFactory scf = SOAPConnectionFactory.newInstance();
        SOAPConnection connection = scf.createConnection();
        // 使用 URLEndpoint 构造接收方地址
        URLEndpoint urlEndpoint = new URLEndpoint(serviceurl);
        // 发送 SOAP 消息到目的地，并返回一个消息
        SOAPMessage response = connection.call(message, urlEndpoint);
        if (response != null) {
            // 输出 SOAP 消息到控制台
            System.out.println("Receive SOAP message from localhost:");
            response.writeTo(System.out);
```

```
        } else {
            System.err.println("No response received from partner!");
        }
        // 关闭连接
        connection.close();
    }
    public static void main(String[] args) throws SOAPException, Exception {
        SenderWithAtt sender = new SenderWithAtt();
        SOAPMessage message = sender.getMessage();
        sender.send(message);
    }
}
```

2. 创建 SOAP 服务器程序（JAXMReceiveServletWithAtt.java），该 Servlet 将在程序内部迭代处理所有消息附件，并根据预存的“文件类型-扩展名”的对应信息（map）将附件写入到本地磁盘。代码如下：

```
package soap;
import java.io.File;
import java.io.FileOutputStream;
import java.util.HashMap;
import java.util.Iterator;
import javax.activation.DataHandler;
import javax.servlet.ServletConfig;
import javax.servlet.ServletException;
import javax.xml.messaging.JAXMServlet;
import javax.xml.messaging.ReqRespListener;
import javax.xml.soap.AttachmentPart;
import javax.xml.soap.MessageFactory;
import javax.xml.soap.SOAPEnvelope;
import javax.xml.soap.SOAPMessage;
public class JAXMReceiveServletWithAtt extends JAXMServlet implements
        ReqRespListener {
    static MessageFactory mf = null;
    private String dir = "c:\\";
    private HashMap<String, String> map;
    // 初始化 MIME 信息与文件扩展名的对照
    public void initMIMEInfo() {
        map = new HashMap<String, String>();
        map.put("image/jpeg", ".jpg");
        map.put("image/gif", ".gif");
    }
    public void init(ServletConfig sc) throws ServletException {
```

```
        super.init(sc);
        // 初始化 MIME 信息
        initMIMEInfo();
        // 创建一个消息工厂
        try {
            mf = MessageFactory.newInstance();
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
    // 处理传过来的 SOAP 消息，并返回一个 SOAP 消息
    public SOAPMessage onMessage(SOAPMessage msg) {
        SOAPMessage resp = null;
        try {
            FileOutputStream fos = null;
            // 循环处理附件
            Iterator attIterator = msg.getAttachments();
            while (attIterator.hasNext()) {
                AttachmentPart attPart = (AttachmentPart) attIterator.next();
                // 获取附件 ID
                String contentId = attPart.getContentId();
                // 获取附件类型
                String contentType = attPart.getContentType();
                System.out.println("Content ID :" + contentId);
                System.out.println("Content Type :" + contentType);
                DataHandler handler = attPart.getDataHandler();
                // 根据附件 ID、附件类型构建要写入的文件名
                String filename = dir + contentId + map.get(contentType);
                // 将附件内容写入文件
                fos = new FileOutputStream(new File(filename));
                handler.writeTo(fos);
                fos.close();
            }
            // 写入文件前，移除所有附件
            msg.removeAllAttachments();
            // 将 SOAP 请求消息写入指定文件
            fos = new FileOutputStream(new File(dir + "soapmessage.xml"));
            msg.writeTo(fos);
            fos.close();
            // 此处省略返回消息处理的业务逻辑，而用硬编码形式创建一个返回消息
            resp = mf.createMessage();
            SOAPEnvelope se = resp.getSOAPPart().getEnvelope();
            se.getBody().addChildElement(se.createName("ResponseMessage"))
```

```
                    .addTextNode("Received Message,Thanks");
            return resp;
        } catch (Exception e) {
            e.printStackTrace();
        }
        return resp;
    }
}
```

3. 修改 web.xml 部署文件以发布该 Servlet，调整后的 web.xml 的内容如下：

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app version="2.4" xmlns="http://java.sun.com/xml/ns/j2ee"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee
    http://java.sun.com/xml/ns/j2ee/web-app_2_4.xsd">
    <servlet>
        <servlet-name>StudentInfoServletWithAtt</servlet-name>
        <servlet-class>soap.JAXMReceiveServletWithAtt</servlet-class>
        <load-on-startup>1</load-on-startup>
    </servlet>
    <servlet-mapping>
        <servlet-name>StudentInfoServletWithAtt</servlet-name>
        <url-pattern>/StudentInfoServletWithAtt</url-pattern>
    </servlet-mapping>
</web-app>
```

4. 部署该 Web 应用，并启动服务器。
5. 执行客户端程序（SenderWithAtt.java），此时在 Web 服务器的控制台将输出附件的相关信息，内容如下：

```
Content ID :yahoo_logo
Content Type :image/jpeg
Content ID :baidu_logo
Content Type :image/gif
```

6. 查看 Servlet 指定的存储目录，会发现附件的内容已接收并被存储。

注意 该服务器 Servlet 在处理完附件后将 SOAP 消息写入到文件，再写入之前务必调用 SOAPMessage 的 removeAllAttachments()方法移除所有附件。



知识拓展

SOAP 安全性

网络的安全性永远是一个重要的问题，SOAP 消息在网络上传输，其安全性在通信期间的访问控制、加密和数据完整性中扮演着一个重要的角色。一般说来，SOAP 无须具备或定义任何特定的安全机制，但使用 SOAP 消息头可以提供一种方法，以基于 XML 的元数据形式定义和添加一些功能，启用应用程序特定的安全性实现方案。元数据信息可以是集成了消息安全性与关联的安全算法（如加密和数字签名）的应用程序特定消息。更重要的是，SOAP 支持将各种传输协议用于通信，因此可以为 SOAP 消息集成 SSL/TLS 等传输协议支持的安全机制。

SOAP 规范中定义了三种安全元素标记：<SOAP-SEC:Encryption>、<SOAP-SEC:Signature>和<SOAP-SEC:Authorization>。使用这些安全标记可在 SOAP 消息中实现加密、数字签名和身份验证。

（1）SOAP 加密

通过在 SOAP 中使用基于 XML 的加密，可对 SOAP 信封中的任何元素进行加密，从而实现安全的通信和访问控制。W3C XML Encryption Work Group（XENC）定义了 SOAP 消息中的 XML 加密。在 SOAP 通信中，加密可以在 SOAP 发送方结点或消息路径中的任何中间方进行。下述的 SOAP 消息使用了 XML 加密来对数据元素进行加密。

```
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope">
    <SOAP-ENV:Header>
        <SOAP-SEC:Encryption
                xmlns:SOAP-SEC="http://schemas.xmlsoap.org/soap/security/"
            SOAP-ENV:actor="http://some URL" SOAP-ENV:mustUnderstand="1">
            <SOAP-SEC:EncryptedData>
                <SOAP-SEC:EncryptedDataReference URI="#encrypted-element" />
            </SOAP-SEC:EncryptedData>
            <xenc:EncryptedKey xmlns:xenc="http://www.w3.org/2001/04/xmlenc#"
                Id="myKey"
                CarriedKeyName="Symmetric Key"
                Recipient="Bill Allen">
                <xenc:EncryptionMethod
                        Algorithm="http://www.w3.org/2001/04/xmlenc#rsa-1_5" />
                <ds:KeyInfo xmlns:ds="http://www.w3.org/2000/09/xmldsig#">
                    <ds:KeyName>Bill Allen's RSA Key</ds:KeyName>
                </ds:KeyInfo>
                <xenc:CipherData>
                    <xenc:CipherValue>ENCRYPTED KEY</xenc:CipherValue>
                </xenc:CipherData>
```

```
            <xenc:ReferenceList>
                <xenc:DataReference URI="#encrypted-element" />
            </xenc:ReferenceList>
        </xenc:EncryptedKey>
    </SOAP-SEC:Encryption>
</SOAP-ENV:Header>
<SOAP-ENV:Body>
    ......
</SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

上述 SOAP 消息中，使用<SOAP-SEC:Encryption>加密了 SOAP 消息头中引用的数据。使用对称密钥加密了<SOAP-SEC:EncryptedData>元素中引用的主体元素，对称密钥在<xenc:EncryptedKey>元素中定义。在 SOAP 接收方结点中，接收方通过关联 DecryptionInfoURI 来解密各个已加密元素，DecryptionInfoURI 指示<xenc:DecryptionInfo>提供如何加密的信息。

（2）SOAP 数字签名

在 SOAP 中使用基于 XML 的数字签名可在通信中提供消息身份验证、完整性和数据的不可否认性（non-repudiation）等功能。发出消息的 SOAP 发送方结点在 SOAP 主体中采用基于 XML 的数字签名，接收方结点则验证该签名。下述的 SOAP 消息使用了 SOAP 数字签名。

```
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope">
    <SOAP-ENV:Header>
        <SOAP-SEC:Signature
                xmlns:SOAP-SEC="http://schemas.xmlsoap.org/soap/security/"
            SOAP-ENV:actor="http://some URL" SOAP-ENV:mustUnderstand="1">
            <ds:Signature Id="TestSignature"
                    xmlns:ds="http:www.w3.org/2000/02/xmldsig#">
                <ds:SignedInfo>
                    <ds:CanonicalizationMethod
                Algorithm="http://www.w3.org/TR/2000/CR-xml-c14n-20001026" />
                    <ds:SignatureMethod
                Algorithm="http://www.w3.org/2000/09/xmldsig#hmac-sha1" />
                    <ds:Reference URI="#Body">
                        <ds:Transforms>
                            <ds:Transform
                Algorithm="http://www.w3.org/TR/2000/CR-xml-c14n-20001026" />
                        </ds:Transforms>
                        <ds:DigestMethod
                        Algorithm="http://www.w3.org/2000/09/xmldsig#sha1" />
                        <ds:DigestValue>some value</ds:DigestValue>
                    </ds:Reference>
```

```
                </ds:SignedInfo>
                <ds:SignatureValue>JHJH2374e</ds:SignatureValue>
            </ds:Signature>
        </SOAP-SEC:Signature>
    </SOAP-ENV:Header>
    <SOAP-ENV:Body>
        ......
    </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

上述 SOAP 消息中，使用<SOAP-SEC:signature>对 SOAP 消息头中所含的数据应用了基于 XML 的数字签名。它使用<ds:CanonicalizationMethod>、<ds:signatureMethod>和<ds:Reference>元素定义算法方法和签名信息。<ds:CanonicalizationMethod>指用于规范化在签名之前摘取（digest）的<signedInfo>元素的算法。<signatureMethod>则定义将规范化的<signedInfo>转换为<signatureValue>的算法。

（3）SOAP 身份验证

SOAP 消息中可使用基于 XML 的身份验证，以来自初始 SOAP 发送方结点的证书对 SOAP 消息进行身份验证。SOAP 身份验证将源于独立的身份验证授权机构、基于 XML 的数字证书应用于来自发送方的 SOAP 消息。下述的 SOAP 消息使用了基于 XML 的身份验证。

```
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope">
    <SOAP-ENV:Header>
        <SOAP-SEC:Authorization
                xmlns:SOAP-SEC="http://schemas.xmlsoap.org/soap/security/"
            SOAP-ENV:actor="http://some URL" SOAP-ENV:mustUnderstand="1">
            <AttributeCert
            xmlns="http://schemas.xmlsoap.org/soap/security/AttributeCert">
                证书
            </AttributeCert>
        </SOAP-SEC:Authorization>
    </SOAP-ENV:Header>
    <SOAP-ENV:Body>
        ......
    </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

上述 SOAP 消息中，使用<SOAP-SEC:Authorization>对 SOAP 消息使用了基于 XML 的身份验证。使用<AttributeCert>元素定义来自独立的授权机构的证书。也可以使用接收方结点或参与方的公钥对其进行加密。在 SOAP 接收结点上，参与方使用其私钥来获取证书。



拓展练习

练习 1.E.1

针对实践 1.G.1 中的 SOAP 消息，写出经过 SOAP 加密、数字签名和身份验证后的消息。

实践 2　CXF 应用



实践指导

实践 2.G.1

搭建 CXF 环境，并创建一个的 Web Service 应用，来模拟电子网站的支付功能。

1. 下载 CXF 框架，并把相应的 jar 包复制到工程中。
2. 基于 CXF 框架，创建 Web Service 服务端和客户端，其中，服务端模拟实现用户的电子支付功能，在客户端打印调用 Web Service 后返回的结果。

分析

1. CXF 旨在为 Service 创建必要的基础设施，它的整体架构主要由 Bus（主干）、Interceptor、前端、服务模型、绑定和传输这几部分，分别介绍如下。
   - **Bus：** Bus 是 CXF 架构的主干，为共享资源提供了一个可配置的场所，作用非常类似于 Spring 的 ApplicationContext。这些共享资源包括 WSDL 管理器、绑定工厂等。通过对 Bus 进行扩展，可以方便地容纳自己的资源或替换现有的资源。默认 Bus 实现是基于 Spring 的，通过依赖注入，将运行时组件组装起来。Bus 的创建由 BusFactory 负责，默认是 SpringBusFactory，对应于默认 Bus 实现。在构造过程中，SpringBusFactory 会搜索 META-INF/cxf（就包含在 CXF 的 Jar 中）下的所有 Bean 配置文件，根据它们构建一个 ApplicationContext。开发者也可提供自己的配置文件来定制 Bus。
   - **消息传递和拦截器（Interceptor）：** CXF 建立于一个通用的消息层之上，主要由消息、拦截器和拦截器链（InterceptorChain）组成。CXF 是以消息处理为中心的，熟悉 JSP/Servlet 的开发者可以将拦截器视为 CXF 架构中的“Filter”，拦截器链也与“FilterChain”类似。通过拦截器，开发者可以方便地在消息传递、处理的整个过程中对 CXF 进行扩展。拦截器的方法主要有两个：handleMessage 和 handleFault，分别对应消息处理和错误处理。在开发拦截器的时候需要注意以下两点。
     - 拦截器不是线程安全的，不建议在拦截器中定义实例变量并使用它。这一点跟 JSP/Servlet 中对于 Filter 的处理是一样的。
     - 不需要调用下一个拦截器的 handleMessage 或 handleFault，这个工作由 InterceptorChain 来完成。

- **前端（Front End）**：它为 CXF 提供了创建服务的编程模型，当前主要的前端就是 JAX-WS。
- **服务模型**：CXF 中的服务通过服务模型来表示。它主要有两部分，即 ServiceInfo 和服务本身。ServiceInfo 作用类似 WSDL，包含接口信息、绑定、端点（EndPoint）等信息；服务则包含了 ServiceInfo、数据绑定、拦截器和服务属性等信息。可使用 Java 类和 WSDL 来创建服务。一般是由前端负责服务创建，它通过 ServiceFactory 来完成。
- **绑定（Binding）**：绑定提供了在传输之上映射具体格式和协议的方法，主要的两个类是 Binding 和 BindingFactory。其中，BindingFactory 负责创建 Binding。
- **传输（Transport）**：为了向绑定和前端屏蔽传输细节，CXF 提供了自己的传输对象。其中主要有两个对象；即 Conduit 和 Destination。前者是消息发送的基础，后者则对应消息接收。开发者还可以给 Conduit 和 Destination 注册 MessageObserver，以便在消息发送和接收时获得通知。

2. CXF 的下载地址是 http://cxf.apache.org/download.html，下载版本为 2.3.2。CXF 可以创建的 Web Service 应用有两种：服务提供者和服务消费者。这种结构与 C/S（客户端/服务器端）结构类似，服务消费者类似于客户端，服务提供者类似于服务器。使用 CXF 创建应用时，服务提供者和服务消费者并不需要同时出现，因为有可能应用只是作为服务提供者或服务消费者单独出现。
3. 默认情况下，CXF 与 Spring 进行了无缝的集成，在本实践练习中，不使用 Spring 框架，CXF 框架提供了 CXFNonSpringServlet 类，用于在非 Spring 框架环境下使用 CXF，这时只要继承该类，并重写该类的 loadBus 方法即可。

参考解决方案

1. 环境搭建

下载版本为 2.3.2 的 CXF 框架，然后把下列包复制到 Web 工程的 lib 文件夹下，如图 2.1 所示。

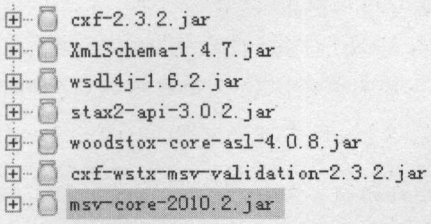


图 2.1　项目所需 jar 包

图 2.1 中，除了“msv-core-2010.2.jar”包需要在 http://wiki.fasterxml.com/WoodstoxDownload 下载之外，其他包均可在“apache-cxf-2.3.2\lib”中找到。

2. 创建 Web 服务发布类

创建一个名为 WSServlet 的类，该类继承了 CXFNonSpringServlet 类，代码如下所示：

```
import javax.servlet.ServletConfig;
import javax.servlet.ServletException;
import javax.xml.ws.Endpoint;
import org.apache.cxf.transport.servlet.CXFNonSpringServlet;
//Web 服务类
public class WSServlet extends CXFNonSpringServlet {
    @Override
    public void loadBus(ServletConfig servletConfig) throws ServletException {
        super.loadBus(servletConfig);
        //发布服务
        Endpoint.publish("/creditcard", new ProcessCreditCard());
    }
}
```

上述代码需要注意以下几点。

- Endpoint 类用来进行服务的发布，Endpoint 还可以连续发布多个服务，其语法格式如下：

```
Endpoint.publish("地址 1",接口 1)
        .publish("地址 2",接口 2)
        .publish("地址 3",接口 3);
```

- “/creditcard”为 web.xml 中的 url-pattern 匹配完毕后剩余的 url。
- ProcessCreditCard 则为 Web 服务接口的实现类。

web.xml 中 WSServlet 的注册代码如下所示：

```
<servlet>
    <!-- Web 服务发布类 -->
    <servlet-name>WSServlet</servlet-name>
    <servlet-class>server.WSServlet</servlet-class>
</servlet>
<servlet-mapping>
    <servlet-name>WSServlet</servlet-name>
    <url-pattern>/ws/*</url-pattern>
</servlet-mapping>
```

3. 创建 Web 服务的接口与实现类

Web 服务接口的代码如下所示：

```
import javax.jws.WebService;
/**
 * Web 服务接口，模拟实现电子支付功能
 *
 * @author haiersoft
 *
 */
@WebService
public interface IProcessCredit {
    /**
     * @param  cardId 信用卡 id
     * @param  total  消费金额
     * @return 处理结果
     */
    public int creditProcess(String cardId,double total);
}
```

上述代码中，WebService 注解表明了 IProcessCredit 接口是一个 Web 服务接口，该接口中的所有方法都被将发布为 Web Service 客户端可以调用的方法。

Web 服务接口实现类的代码如下所示：

```
import java.util.HashMap;
import java.util.Map;
import javax.jws.WebService;
/**
 * 信用卡处理接口的实现类
 *
 * @author haiersoft
 *
 */
@WebService(endpointInterface = "server.IProcessCredit")
public class ProcessCreditCard implements IProcessCredit {
    // 存储信用卡号和余额，模拟数据库
    private static Map<String, Double> data = new HashMap<String, Double>();
    /**
     * @param cardId  信用卡 id
     * @param total   消费金额
     * @return 处理结果
     */
    @Override
    public int creditProcess(String cardId, double total) {
        initData();// 如果 Map 还没有初始化，进行初始化
        // 判断是否有此卡号
```

```
        if (!data.containsKey(cardId)) {
            return -1;// 没有此卡号
        }
        double balance = data.get(cardId);// 返回余额
        if (total < balance) {
            return 0;// 余额不足
        }
        balance = balance - total;// 减去余额
        data.put(cardId, balance);// 重新设置
        // 成功返回
        return 1;
    }
    /**
     * 初始化 Map 结构
     *
     * @return
     */
    void initData() {
        if (data.size() == 0) {
            // 初始化数据
            data.put("12345", 1000d);
            data.put("23456", 1002d);
            data.put("34567", 10000d);
            data.put("45678", 1000d);
        }
    }
}
```

上述代码中，通过使用 WebService 注解中的 endpointInterface 属性明确指出了该实现类的接口是“server.IProcessCredit”。WebService 注解中的属性见表 2.1。

表 2.1 WebService 的元素

元素	内容描述
endpointInterface	定义服务抽象 Web Service 协定的服务端点接口的完整名称
name	Web Service 的名称
portName	Web Service 的端口名称
serviceName	Web Service 的服务名称
targetNamespace	如果@WebService.targetNamespace 注释是关于某一服务端点接口的，则 targetNamespace 用于 wsdl:portType（以及关联的 XML 元素）的名称空间
wsdlLocation	描述服务的预定义 WSDL 的位置

启动 Tomcat 服务器后，访问：

```
http://localhost:8080/ph02/ws/creditcard?wsdl
```

结果如图 2.2 所示。

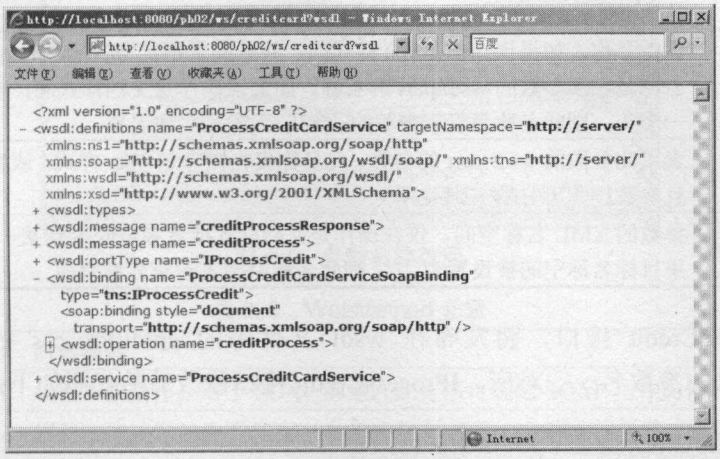


图 2.2　访问效果图

图 2.2 中显示了 Web Service 的 WSDL 已经生成，说明 Web Service 发布成功。

在浏览器中输入：

```
http://localhost:8080/ph02/ws/creditcard/creditProcess?arg0=1234567&arg1=300
```

返回结果如图 2.3 所示。

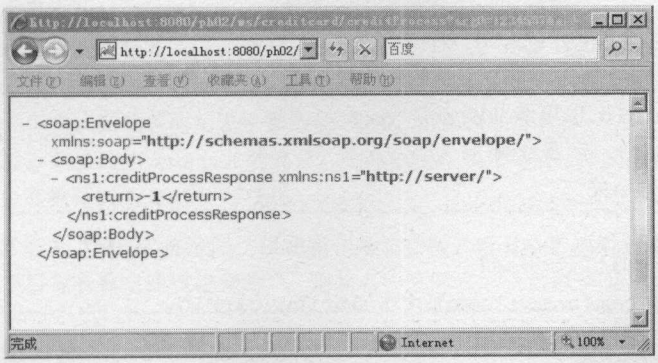


图 2.3 访问效果图

上述 URL 中的 creditProcess 与接口中的 ProcessCreditCard 方法对应，arg0 则与 creditProcess 方法的参数 cardId 对应。arg1 与方法的参数 total 对应，arg0 和 arg1 都是系统自动生成的名称，如果指定一个有意义的参数名称，可以使用 WebParam 注解，关于 WebParam 中的元素，见表 2.2。

表 2.2　WebParam 元素

元素	内容描述
header	如果为 true，则参数是从消息头而不是消息正文获取的

（续表）

元素	内容描述
mode	该元素为 WebParam.Mode 类型，表示参数的流向（IN、OUT 或 INOUT 之一）。默认为 IN
name	参数名称，如果该操作是 rpc 样式的，并且尚未指定 @WebParam.partName，则此名称是表示参数的 wsdl:part 的名称。如果该操作是文档样式的，或者参数映射到某一个头，则此名称是表示参数的 XML 元素的本地名称
partName	表示此参数的 wsdl:part 的名称。此名称仅在操作是 rpc 样式，或者操作是文档样式且参数样式为 BARE 时使用
targetNamespace	参数的 XML 名称空间。仅在操作是文档样式或者参数映射到某一个头时使用。如果目标名称空间被设置为 ""，则此名称空间表示空名称空间

修改 IProcessCredit 接口，将发布在 wsdl 文件中的 creditProcess 的参数与接口中 creditProcess 方法的参数名一一对应。IProcessCredit 接口修改后的代码如下：

```
import javax.jws.WebParam;
import javax.jws.WebService;
/**
 * Web 服务接口，模拟实现电子支付功能
 *
 * @author haiersoft
 *
 */
@WebService
public interface IProcessCredit {
    /**
     * @param  cardId 信用卡 id
     * @param  total  消费金额
     * @return 处理结果
     */
    public int creditProcess(
            @WebParam(name="cardId") String cardId,
            @WebParam(name="total")  double total);
}
```

这时，发布后的参数名称由 arg0 变为 cardId，arg1 变为 total，如图 2.4 所示。

除了 WebParam 和 WebService 注解之外，还有以下几个重要的注解，分别是 WebMethod、WebResult。其中，WebMethod 定制一个公开为 Web Service 操作的方法，关联方法必须是公共方法且其参数可以返回值，并且异常必须遵循 JAX-RPC 1.1 的第 5 节中定义的规则。该方法不需要抛出 java.rmi.RemoteException，其元素介绍见表 2. 3。

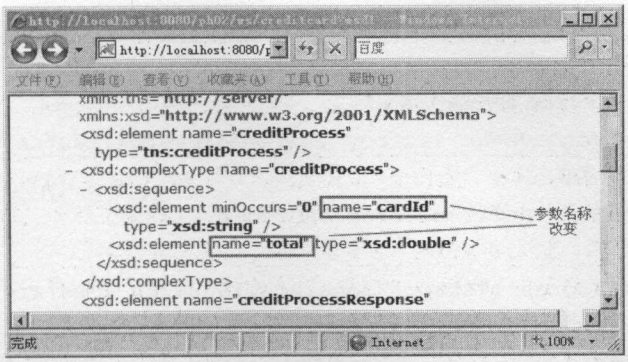


图 2.4　参数名称改变

表 2.3　WebMethod 元素

元素	内容描述
action	此操作的动作。对于 SOAP 绑定，此方法可确定 SOAP 动作的值
exclude	将某一方法标记为不作为一个 Web 方法公开。此方法用于阻止将某一继承方法公开为此 Web 服务的一部分
operationName	与此方法匹配的 wsdl:operation 的名称

WebResult 注解主要用于定制返回值到 WSDL 部分和 XML 元素的映射关系。其元素介绍见表 2.4。

表 2.4　WebResult 元素

元素	内容描述
header	如果为 true，则参数是从消息头而不是消息正文获取的
name	参数名称，如果该操作是 rpc 样式的，并且尚未指定 @WebParam.partName，则此名称是表示参数的 wsdl:part 的名称。如果该操作是文档样式的，或者参数映射到某一个头，则此名称是表示参数的 XML 元素的本地名称
partName	表示此参数的 wsdl:part 的名称。此名称仅在操作是 rpc 样式，或者操作是文档样式且参数样式为 BARE 时使用
targetNamespace	参数的 XML 名称空间。仅在操作是文档样式或者参数映射到某一个头时使用。如果目标名称空间被设置为""，则此名称空间表示空名称空间

4. 创建 Web 服务客户端

Web 服务客户端的代码如下：

```
import org.apache.cxf.jaxws.JaxWsProxyFactoryBean;
import server.IProcessCredit;
/**
 * Web服务客户端
 * @author haiersoft
 *
 */
public class Client {
```

```
    public static void main(String[] args) {

        // 创建 WebService 客户端代理工厂
        JaxWsProxyFactoryBean factory = new JaxWsProxyFactoryBean();
        // 设置 Web 服务的接口
        factory.setServiceClass(IProcessCredit.class);
        // 设置地址
        factory.setAddress("http://localhost:8080/ph02/ws/creditcard");
        // 通过代理工厂返回接口的实例
        IProcessCredit client = (IProcessCredit) factory.create();
        // 传入参数
        int result = client.creditProcess("23456", 1500);
        if (result == -1) {
            System.out.println("信用卡号不正确!");
        }
        if (result == 0) {
            System.out.println("余额不足! ");
        }
        if (result == 1) {
            System.out.println("信用卡支付成功! ");
        }
    }
}
```

运行上述代码后，结果如下：

```
信用卡支付成功!
```



知识拓展

Web Service 框架

Java 领域的 Web Service 框架众多，下面简要介绍在开发过程中常用的其他 Web 服务框架。

1. Axis

Axis（Apache EXtensible Interaction System）本质上就是一个 SOAP 引擎，提供创建服务器端、客户端和网关 SOAP 操作的基本框架。Axis 目前版本是为 Java 编写的，不过为 C++ 编写的版本正在开发中。除了提供 SOAP 引擎功能外，Axis 还是一个独立的 SOAP 服务器和一个嵌入 Servlet 引擎（例如 Tomcat）的服务器。

从 2000 年起，SOAP v2 开发小组开始讨论如何让 Axis 更加灵活，可配置，以及能够处

理 SOAP 和来自 W3C 的各种 XML 标准。通过不断地讨论和代码编写，Axis 目前取得了如下成果。

- 速度提高。Axis 通过基于事件的 SAX 对 XML 文档进行处理，从而在速度和效率上比 Apache SOAP 有所提高。
- 灵活性提高。
- 稳定性提高。
- 提供面向组件的部署。
- 提供一个简洁的传输抽象框架。其核心引擎完全于传输方式独立。从而使基于何种协议传输的选择更加灵活。
- 支持 WSDL。包括导出 WSDL 和客户代理生成等。

要了解详情，参见：

```
http://axis.apache.org
```

2. Axis 2

Axis 2 是 Apache Axis 的下一代。Axis 2 虽然由 Axis1.x 处理程序模型提供支持，但它具有更强的灵活性并可扩展到新的体系结构。Axis 2 基于新的体系结构进行了全新编写，而且没有采用 Axis 1.x 的常用代码。支持开发 Axis 2 的动力是探寻模块化更强、灵活性更高和更有效的体系结构，这种体系结构可以很容易地插入到其他相关 Web 服务标准和协议（如 WS-Security、WS-ReliableMessaging 等）的实现中。Axis 2 的主要特点如下。

- 采用名为 AXIOM（AXIs Object Model）的新核心 XML 处理模型，利用新的 XML 解析器提供的灵活性按需构造对象模型。
- 支持不同的消息交换模式。目前 Axis 2 支持三种模式：In-Only、Robust-In 和 In-Out。In-Only 消息交换模式只有 SOAP 请求，而不需要应答；Robust-In 消息交换模式发送 SOAP 请求，只有在出错的情况下才返回应答；In-Out 消息交换模式总是存在 SOAP 请求和应答。
- 提供阻塞和非阻塞客户端 API。
- 支持内置的 Web 服务寻址(WS-Addressing)。
- 灵活的数据绑定，可以选择直接使用 AXIOM，使用与原来的 Axis 相似的简单数据绑定方法，或使用 XMLBeans、JiBX 或 JAXB 2.0 等专用数据绑定框架。
- 新的部署模型，支持热部署。
- 支持 HTTP，SMTP，JMS，TCP 传输协议。
- 支持 REST 架构（Representational State Transfer，表述性状态转移）。

要了解详情，参见：http://axis.apache.org。

3. XFire

XFire 是 codeHaus 组织提供的一个开源框架，它构建了 POJO 和 SOA 之间的桥梁，主要特性就是支持将 POJO 通过非常简单的方式发布成 Web 服务，这种处理方式不仅充分发挥了 POJO 的作用，简化了 Java 应用转化为 Web 服务的步骤和过程，也直接降低了 SOA 的实现难度，为企业转向 SOA 架构提供了一种简单可行的方式。该框架具有以下特点。

- 支持将 Web 服务绑定到 POJO、XMLBeans、JAXB1.1、JAXB2.0 和 Castor。
- 支持基于 HTTP、JMS、XMPP 等多种协议访问 Web 服务。
- 支持多种 Web 服务业界重要标准如 SOAP、WSDL、Web 服务寻址（WS-Addressing）、Web 服务安全（WS-Security）等。
- 支持 JSR181，可以通过 JDK5 配置 Web 服务。
- 高性能的 SOAP 实现。
- 服务器端、客户端代码辅助生成。
- 对 Spring、Pico、Plexus 等项目的支持等。

要了解详情，可参见：http://xfire.codehaus.org。



拓展练习

练习 2.E.1

查阅 XFire 的使用方法，把 CXF 使用 XFire 代替，完成电子支付的功能。

附录 A　XSD 元素

元素	描述
all	规定子元素能够以任意顺序出现，每个子元素可出现零次或一次
annotation	annotation 元素是一个顶层元素，规定 schema 的注释
any	使创作者可以通过未被 schema 规定的元素来扩展 XML 文档
anyAttribute	使创作者可以通过未被 schema 规定的属性来扩展 XML 文档
appInfo	规定 annotation 元素中应用程序要使用的信息
attribute	定义一个属性
attributeGroup	定义在复杂类型定义中使用的属性组
choice	仅允许在〈choice〉声明中包含一个元素出现在包含元素中
complexContent	定义对复杂类型（包含混合内容或仅包含元素）的扩展或限制
complexType	定义复杂类型
documentation	定义 schema 中的文本注释
element	定义元素
extension	扩展已有的 simpleType 或 complexType 元素
field	规定 XPath 表达式，该表达式规定用于定义标识约束的值
group	定义在复杂类型定义中使用的元素组
import	向一个文档添加带有不同目标命名空间的多个 schema
include	向一个文档添加带有相同目标命名空间的多个 schema
key	指定属性或元素值（或一组值）必须是指定范围内的键
keyref	规定属性或元素值（或一组值）对应指定的 key 或 unique 元素的值
list	把简单类型定义为指定数据类型的值的一个列表
notation	描述 XML 文档中非 XML 数据的格式
redefine	重新定义从外部架构文件中获取的简单和复杂类型、组和属性组
restriction	定义对 simpleType、simpleContent 或 complexContent 的约束
schema	定义 schema 的根元素
selector	指定 XPath 表达式，该表达式为标识约束选择一组元素
sequence	要求子元素必须按顺序出现。每个子元素可出现 0 到任意次数
simpleContent	包含对 complexType 元素的扩展或限制且不包含任何元素
simpleType	定义一个简单类型，规定约束以及关于属性或仅含文本的元素的值的信息
union	定义多个 simpleType 定义的集合
unique	指定属性或元素值（或者属性或元素值的组合）在指定范围内必须是唯一的

附录 B　XSD 限定

限定	描述
enumeration	定义可接受值的一个列表
fractionDigits	定义所允许的最大的小数位数。必须大于等于 0
length	定义所允许的字符或者列表项目的精确数目。必须大于或等于 0
maxExclusive	定义数值的上限。所允许的值必须小于此值
maxInclusive	定义数值的上限。所允许的值必须小于或等于此值
maxLength	定义所允许的字符或者列表项目的最大数目。必须大于或等于 0
minExclusive	定义数值的下限。所允许的值必须大于此值
minInclusive	定义数值的下限。所允许的值必须大于或等于此值
minLength	定义所允许的字符或者列表项目的最小数目。必须大于或等于 0
pattern	定义可接受的字符的精确序列
totalDigits	定义所允许的阿拉伯数字的精确位数。必须大于 0
whiteSpace	定义空白字符（换行、回车、空格以及制表符）的处理方式

附录 C　XPath 函数库

结点函数

函数	返回值类型	返回值
position()	数值	上下文结点列表中上下文结点的位置。列表中的第一个结点其位置为 1
last()	数值	上下文结点集合中的结点数
count()	数值	统计计数，返回符合条件的结点的个数
name()	字符串	返回元素的名称
id(string)	结点集合	结点集合，其中只有一个元素（在同一个文档的任何位置）其 ID 为 string
idref(node-set)	结点集合	结点集合，包括文档中的某些元素，其 ID 属性为在参数 node-set 中指明结点值中的（以空格分开）记号之一
keyref(string name, node set values)	结点集合	结点集合，包括文档中所有具有某种关键字结点，此关键字的值与第二个参数中的结点值相同
doc(string URI)	结点集合	文档或由 URI 引用的文档部分中的结点集合；这些结点从 URI 使用的命名的 anchor 标记或 XPointer 中选择。如果没有命名的 anchor 标记或 Xpointer，那么所指文档的根元素就存在于结点集合中。相对 URI 是相对于输入文档中的当前结点的
docref(node set)	结点集合	结点集合，包括由 URI 引用的，其值为 node set 参数的所有结点
local-part(node set)	字符串	node set 参数中第一个结点的本地部分（命名域前缀后面的所有内容）；当不使用任何参数时可用于获取上下文结点的本地部分
namespace(node set)	字符串	结点集合中第一个结点命名域的 URI；当不使用任何参数时，可用于获得上下文结点的命名域 URI；如果结点处于默认命名域内，则返回空字符
qname(node set)	字符串	node set 参数中第一个结点的合法名称(可以为前缀和本地部分)；要获得上下文结点的合法名称，可不使用任何参数
generate-id(node set)	字符串	参数 node set 中第一个结点的唯一标识符；不带参数使用时，可生成上下文结点的 ID

字符串函数

函数	作用
concat()	将参数字符串连接后返回
contains()	如果第一个字符串包含第二个字符串，则返回真，否则返回假
normalize-space()	将字符串中前后空格删除后返回
starts-with()	如果第一个字符串以第二个字符串开头则返回真，否则返回假
string()	将一个对象转换为字符串后返回
string-length()	返回一个字符串的长度

（续表）

函数	作用
substring()	从字符串的某个偏移位置开始返回指定长度的字符串
substring-after()	返回字符串中从开始到某一字符串第一次出现的位置后的字符串
substring-before()	返回字符串中从开始到某一字符串第一次出现的位置前的字符串
translate()	替换一个字符串中字符

数值函数

函数	作用
ceiling()	返回不小于参数的最小整数
floor()	返回不大于参数的最大整数
number()	转换参数为一个数字
round()	对参数四舍五入
sum()	返回所有参数的和

布尔函数

函数	作用
boolean()	将参数转换为布尔值
false()	返回假
not()	如果参数是假，则返回真，否则返回假
true()	返回真

附录 D　XSL 元素

元素	功能描述
apply-imports	应用来自导入样式表中的模版规则
apply-templates	向当前元素或当前元素的子元素应用模板
attribute	向元素添加属性
attribute-set	创建命名的属性集
call-template	调用一个指定的模板
choose	与<when>以及<otherwise>协同使用，来表达多重条件测试
comment	在结果树中创建注释结点
copy	创建当前结点的一个备份（无子结点及属性）
copy-of	创建当前结点的一个备份（带有子结点及属性）
decimal-format	定义当通过 format-number() 函数把数字转换为字符串时，所要使用的字符和符号
element	在输出文档中创建一个元素结点
fallback	假如处理器不支持某个 XSLT 元素，规定一段备用代码来运行
for-each	遍历指定的结点集中的每个结点
if	包含一个模板，仅当某个指定的条件成立时应用此模板
import	用于把一个样式表中的内容倒入另一个样式表中
include	把一个样式表中的内容包含到另一个样式表中
key	声明一个命名的键
message	向输出写一条消息（用于错误报告）
namespace-alias	把样式表中的命名空间替换为输出中不同的命名空间
number	测定当前结点的整数位置，并对数字进行格式化
otherwise	规定 <choose> 元素的默认动作
output	定义输出文档的格式
param	声明一个局部或全局参数
preserve-space	用于定义保留空白的元素
processing-instruction	生成处理指令结点
sort	对结果进行排序
strip-space	定义应当删除空白字符的元素
stylesheet	定义样式表的根元素
template	当指定的结点被匹配时所应用的规则
text	通过样式表生成文本结点
transform	定义样式表的根元素
value-of	提取选定结点的值
variable	声明局部或者全局的变量
when	规定 <choose> 元素的动作，用于代表某种条件
with-param	规定需被传入某个模板的参数的值

Broadview®
www.broadview.com.cn
博文视点·IT出版旗舰品牌
技术凝聚实力·专业创新出版

高等院校软件专业方向 系列教材

- 《Java SE程序设计基础教程》
- 《Java SE程序设计高级教程》
- 《数据库原理与应用（Oracle版）》
- 《Web编程基础》
- 《Java Web程序设计》
- 《VB.NET程序设计》
- 《ASP.NET程序设计（C#版）》
- 《Java EE轻量级解决方案——S2SH》
- 《Web Services 应用开发》
- ……







上架建议：程序设计

ISBN 978-7-121-13545-3

9 787121 135453 >

定价：39.00元

责任编辑：张月萍
封面设计：李　玲

本书贴有激光防伪标志，凡没有防伪标志者，属盗版图书。